In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

University of Alberta

TaMeX: An Intelligent-Agent Framework for Flexible Service Integration on the Web

by

P. Maurine Hatch ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements of the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta

Fall 2003

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisisitons et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-87776-0
Our file Notre référence
ISBN: 0-612-87776-0

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

Canada

University of Alberta
Library Release Form

Name of Author: P. Maurine Hatch
Title of Thesis: TaMeX: An Intelligent-Agent Framework for Flexible Service Integration on the Web
Degree: Master of Science
Year this Degree Granted: 2003

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Date: June 24, 2003

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **TaMeX: An Intelligent-Agent Framework for Flexible Service Integration on the Web** submitted by **P. Maurine Hatch** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dr. Marek Reformat
Electrical and Computer Engineering

Date: June 23, 2003

ABSTRACT

A rapidly increasing number of services is available on the World Wide Web, which has given rise to a great challenge: to enable the interoperation of these services in the context of high-quality, integrated applications, providing customized value-added services to the end user.

TaMeX is an XML-based intelligent-agent framework that supports the development of agent-based applications, integrating services of pre-existing web applications with newly developed functionalities. The TaMeX framework architecture is based on a set of declarative models of the application-domain information, the supported tasks, the end-user profiles, and the pre-existing web services. At run-time, the TaMeX agents use these models to flexibly interact with the users, monitor and control the execution of the underlying applications and coordinate the information exchange among them, and to collaborate in order to react to failures and effectively accomplish the desired user request.

To my loving family

Acknowledgements

I extend my sincere thanks to my supervisor, Dr. Eleni Stroulia, for her invaluable feedback, guidance, and support throughout this work. Her encouragement, enthusiasm and dedication to my program and future motivated me even when the going got tough.

I thank Gina Situ for her work on the first version of TaMeX and for her patience and assistance in familiarizing me with TaMeX. I thank Eddie Zadrozny for being such a joy to work with and for his substantial contribution to the implementation of the second version. I am also grateful to Roland Penner and Paul Iglinski for their excellent ideas and technical support in the development of the system.

I would like to express my appreciation to every member of the Software Engineering Research group, each one of whom offered helpful advice and assistance and made my graduate studies more enjoyable. Special thanks goes to Mohammad El-Ramly who shared an office with me and was always willing to help and give suggestions. Special thanks also goes to Kavita Gandhi (Jari), Yiqiao Wang, Milli Ambe, and Ying Liu, who not only contributed to my enjoyment of the lab experience, but also the enjoyment of life outside the lab!

Finally, I thank my family for their constant love and support. They encouraged me and could be counted on to uplift my spirits and my motivation whenever I needed it.

P. Maurine Hatch
June 2003.
Edmonton, Canada.

Table of Contents

CHAPTER 1 INTRODUCTION AND MOTIVATION 1

1.1 THE RESEARCH PROBLEM 1

1.2 THE PROPOSED APPROACH 2

1.3 THE CONTRIBUTIONS 3

1.4 THESIS OVERVIEW 4

CHAPTER 2 TAMEX INTELLIGENT-AGENT FRAMEWORK 6

2.1 BOOK-FINDING: AN INTEGRATED-APPLICATION EXAMPLE 7

2.2 THE INTEGRATION-SPECIFICATION LANGUAGE 8

2.2.1 The Domain Model *10*

Meta-model 10

Model 12

User-Session Data 13

2.2.2 The Task Model *13*

Meta-model 13

Model 18

User-Session Data 19

2.2.3 Constraints *19*

2.2.4 The Profile Model *21*

Meta-model 23

Model 24

User-Session Data 25

2.3 THE ARCHITECTURE 25

2.3.1 The Individual Task Agent *25*

2.3.2 The Main Configuration Model *29*

2.3.3 The User Interface Model *33*

2.4 BOOK-FINDING MULTI-AGENT APPLICATION 34

CHAPTER 3 THE TASK AGENT 37

3.1 CONFIGURABLE TASK-MODEL INTERPRETATION 38

3.1.1 Task Model Selection 38
3.1.2 Task Model Interaction 39
3.1.3 Behavior Specialization 40
3.2 PROFILE-BASED CUSTOMIZATION 41
3.3 USER-SESSION STATE MAINTENANCE 42
3.4 REFLECTIVE MONITORING, FAILURE DETECTION AND INTER-AGENT COLLABORATION 43
CHAPTER 4 APPLICATION DEVELOPMENT WITH TAMEX **44**
4.1 DOMAIN MODEL DEVELOPMENT 46
4.2 TASK MODEL DEVELOPMENT 50
4.3 PROFILE MODEL DEVELOPMENT 52
4.4 WRAPPER CONSTRUCTION 52
4.5 TASK AGENT CONFIGURATION 54
4.6 MODEL VALIDATION 57
CHAPTER 5 RUN-TIME BEHAVIOR **60**
5.1 TASK MODEL SELECTION 61
5.2 SELECTED TASK MODEL START-UP 61
5.3 INPUT TASK EXECUTION 62
5.4 WRAPPER TASK EXECUTION 63
5.5 INTERNAL TASK EXECUTION 64
5.6 OUTPUT TASK EXECUTION 65
5.7 USER PROFILE CUSTOMIZATION OF THE TASK MODEL 66
5.8 TASK AGENT-DRIVEN EXECUTION OF THE CUSTOMIZED TASK MODEL 69
5.9 COLLABORATION WITH ANOTHER TASK AGENT 71
5.10 USER-DRIVEN EXECUTION OF THE CUSTOMIZED TASK MODEL 73
5.11 COLLABORATION WITH ANOTHER TASK AGENT UPON FAILURE 73
CHAPTER 6 EVALUATION AND COMPARISON TO RELATED WORK **75**

6.1 Comparisons to Related Work 75

6.1.1 First Version of TaMeX 75
6.1.2 Intelligent Integration Architectures 80
6.1.3 Intelligent Integration Languages 82
6.1.4 Web Services 84

6.2 Case Studies 87

6.2.1 Book-Finding Prototype (Comparative-shopping) 87
6.2.2 Pharmacy Shopping Prototype (Comparative-shopping) 87
6.2.3 Surgery 446 Clinical Reporting Prototype (Logbook Recording) 88

CHAPTER 7 CONCLUSIONS 91

7.1 Contributions 91

7.1.1 Integration-Specification Language 91
7.1.2 Intelligent-agent Run-time Execution Environment 92
7.1.3 User-based Run-time Customization 92
7.1.4 Design-time Toolkit 93

7.2 Future Work 93

7.2.1 Web Services Standards Integration 93
7.2.2 Support for Task-Structure Composition 95
7.2.3 Interaction Experience Integration 95

BIBLIOGRAPHY 96

APPENDICES 101

A - Domain Model (including Profile Model adaptations) 102

1. Domain Model Template for comparative shopping - XML Structure Diagram 102
2. Domain Model Template for comparative shopping - XML Schema 103
3. Domain Model Schema for Book Finding - XML Structure Diagram 108
4. Domain Model Schema for Book Finding - XML Schema 109
5. Price Conceptual Constraint-checking Stylesheet 121

B - Task Model (including Profile Model adaptations) 122

1. Task Model Schema - XML Schema 122
2. Task Model Instance for Book Finding - XML Schema Instance 133
3. Filter-by-Price Task Functional Constraint-checking Stylesheet 152

C - PROFILE MODEL 153

1. Profile Adaptation Stylesheet for Generating the Input Form *153*
2. Profile Adaptation Stylesheet for Automating a Task *159*
3. Profile Adaptation Stylesheet for Removing a Task *161*

D - SYSTEM CONFIGURATION 163

1. Main Configuration - XML Schema *163*
2. Main Configuration Instance - XML Schema Instance *167*

E - USER INTERFACE CONFIGURATION 172

1. Main Display - XML Schema *172*
2. Main Display - XSL Stylesheet for Generating User Interface *175*

F - VALIDATION 179

1. Strict Validation Stylesheet *179*

Index of Tables

Table 1: Integrated Application Issues 6

Table 2: Four Layered Integration-Specification Language - see Appendices A through C 10

Table 3: Constraints used by Reflective Monitoring in our Framework 20

Table 4: User Profile Customization Types 21

Table 5: User Profile Customization of the Book-finding Application 22

Table 6: Main Configuration Model Categories and Elements 30

Table 7: Agent Registry (registeredAgents) 30

Table 8: Task Model Registry (registeredTaskModels) 31

Table 9: Wrapper Registry (registeredResources) 31

Table 10: Important Main Configuration Options for the Book Finding Application 56

Table 11: Strict Validation Rules 59

Table 12: Instructor Profile Customization of the Book-finding Application 69

Index of Figures

Figure 1: Domain Model Template UML Diagram for "comparative shopping" *11*

Figure 2: Task Model Schema UML Diagram *14*

Figure 3: Task Model Schema - XML Structure Diagram *16*

Figure 4: "Autogen" Detail of the Task Model Schema - XML Structure Diagram *17*

Figure 5: Task Structure for the Book-Finding Application *19*

Figure 6: Profile Part of Domain Model Template for "comparative shopping" *24*

Figure 7: The Architecture for a Task Agent *26*

Figure 8: The Detailed Component Diagram for a Task Agent *27*

Figure 9: Main Configuration - XML Structure Diagram *32*

Figure 10: Main Display Schema - XML Structure Diagram *33*

Figure 11: The Component Architecture of our Framework for Book-Finding. *35*

Figure 12: Task Model Selection Screen *38*

Figure 13: Task Model Interaction Screen *39*

Figure 14: Adapt Task Model Screen *41*

Figure 15: TaMeX Development Activities *45*

Figure 16: TaMeX Domain Model Development Activities *46*

Figure 17: Concept Book in Book Finding Domain Model Schema *47*

Figure 18: Book-Finding Domain Model Schema UML Diagram *48*

Figure 19: TaMeX Task Model Development Activities *50*

Figure 20: TaMeX Profile Model Development Activities *52*

Figure 21: TaMeX Wrapper Construction Activities *53*

Figure 22: TaMeX Task Agent Configuration Development Activities *54*

Figure 23: TaMeX Model Validation Development Activities *57*

Figure 24: Task Model Selection Screen *61*

Figure 25: First Task Model Interaction Screen *62*

Figure 26: Specify Book Selection Criteria Screen 63
Figure 27: Access Amazon Screen (after task completed) 63
Figure 28: Sort by Price Screen (after task completed) 65
Figure 29: Show Explorer Applet Screen 66
Figure 30: Instructor Filled-in Adapt Task Model Screen (last portion) 68
Figure 31: Instructor-Customized Hierarchical Task Menu 70
Figure 32: Task Agent-Driven Execution of the Customized Task Model Results Screen 72
Figure 33: Group by Origin Screen (after task completed) 73
Figure 34: Pharmacy Shopping Results Sorted by Price 88
Figure 35: Surgery 446 Desktop Version of Edit All Entries (S100) of the Seminars 90
Figure 36: Surgery 446 Mobile Device Version of Edit All Entries (S100) 90
Figure 37: Web Services Component Diagram for a Task Agent 93

Chapter 1 Introduction and Motivation

1.1 The Research Problem

The World Wide Web today presents a great opportunity and a difficult challenge: it offers a wide and growing variety of potentially useful services but it offers limited support for discovering, selecting and combining them. Users who need to accomplish complex tasks, such as comparing related information from different sources, need first to access multiple web sites to decide which are most likely to be useful. Then, they need to input the information required by the selected web-based applications in the format each one requires, quite often translating on the fly, because these formats are superficially different though part of the input information is the same. Finally, they have to collect, again translate, interpret and combine the web applications' responses, in order to infer the information they need or to complete their task.

It would be much easier if users of complex tasks could simply request the service and have the final solution returned to them instead of executing all the necessary steps themselves. This requires the development of an application that combines the functionalities of many independently developed web-based applications. This integrated application would need to control the information flow of the overall task, translate the information among the existing applications, and combine the many responses received into a final solution.

The current lack of web-based application-interoperation support [Wie95, Wie97] has motivated recent work on the standardization of a stack of related protocols for service discovery, selection and integration [Web]. However, until these standards mature and become widely supported, the application selection and composition will have to be delegated to the end user, or the portal designer. Even when new standardized services become the norm, preexisting services will still need to be either re-implemented or automatically reengineered to meet the standards. Thus, the problem of providing automated support towards integrating existing web applications constitutes a great opportunity for artificial-intelligence research.

1.2 The Proposed Approach

This thesis is aimed towards addressing this challenge of integrating web-based applications. It is part of the TaMeX (**Ta**sk-based **Me**diation through **X**ML) project [SS00, STS00]. TaMeX is a software framework supporting the development of intelligent multi-agent applications that integrate existing web-based applications offering related services in a common domain. The research objectives of the TaMeX project are:

- to develop a suite of intelligent methods for adapting existing web-based applications so that they "speak" the same - in terms of syntax and semantics - language;
- to provide a language with well-defined semantics for specifying the integrated application domain, the pre-existing services, the supported services and their composition;
- to develop a flexible and robust run-time environment for executing the integrated applications and delivering the desired services; and
- to develop methods for run-time user customization of the service composition.

The first version of the TaMeX framework [SS00, STS00] focused mainly on the first issue, namely the (semi-) automatic adaptation of existing web-based applications. This is made possible through the construction of *wrappers* that encapsulate the behavior of the existing web-based applications with a special API. The purpose of this API is to translate the original, HTML-based communication protocols of the existing web-based applications into a common protocol, based on a set of XML-defined data.

This thesis extends the first version of the TaMeX framework by focusing on the other three important issues of the web-application integration problem: to design an expressive well-defined language for specifying domain-specific integration, to develop a flexible and robust run-time environment for executing them, and to develop a means for the users to adapt the system at run-time [HS02].

We have adopted an implementation based on the eXtensible Markup Language [XML] and its related languages [XSD, XSL, XSLT] to provide the syntax for these problems and a consistent artificial-intelligence stance to provide the semantics. An XML-based implementation was chosen since it has many tools supporting its use and is expected to be the basis of Web interoperations. The TaMeX integration-specification language is based on a planning language, the SBF-TMK functional representation language [Cha89, SG99] for specifying the internal processing of intelligent agents. This language enables the declarative specification of workflows, as hierarchically decomposed task structures. The run-time execution environment is based on a distributed intelligent-agent architecture. Agents in this architecture are able to reflectively interpret task structures specified in the above language. The interpretation process results in the coherent interaction of the integrated-application's end user with the wrapped web-based applications. The agents' reflective capability enables them to monitor the interpretation process and to detect failures as violations of the semantic constraints of the integration specification. In response to such failures, the agents employ the capabilities of other agents in the architecture and collaborate to address the end-user's request. Finally, the agents are able to flexibly adapt the integration specifications to individual user's preferences.

1.3 The Contributions

The main contributions of this thesis to the work on the TaMeX intelligent-agent framework are as follows.

1. It offers an extensible integration-specification language that is an XML-based language for declaratively modeling the application-domain information, the supported tasks, the semantic constraints of the domain concepts and tasks, and the user preferences regarding task composition and domain concept values of an integrated application.

2. It provides a flexible, adaptive, distributed environment for run-time execution based on intelligent task agents which use the declarative models specified in the integration-specification language to flexibly interact with the users, monitor and control the execution of the integrated application, and to recognize failures and then collaborate with each other to attempt error recovery and successful completion of the user's task.
3. It supports user-based, run-time customization of the integration specifications. To accomplish this, it provides a profile model integration-specification language for specifying the customizations available and methods for the users to employ at run-time to adapt the task-structure model to their own individual preferences.
4. It provides a design-time toolkit which is a collection of templates and schemas to support the developer in creating the declarative models for an integrated application and a corresponding set of validating tools to ensure that the developed models conform to the rules.

1.4 Thesis Overview

This thesis is organized as follows. Chapters 2 and 3 describe the building blocks of a TaMeX integrated application. Chapter 2 describes the integration-specification language and the architecture of the TaMeX intelligent-agent framework. A book-finding example is described that is used throughout the thesis to illustrate the development and implementation of TaMeX integrated applications. Chapter 3 describes the functionalities of the TaMeX task agent, including the interpretation of the task model, the maintenance of the overall state of this task-model interpretation, the customization of the standard task model and the subsequent interpretation of this user-configured model and the user interaction support. As well, it describes how the task agent monitors the progress of the run-time execution, recognizes invalid information as a failure and attempts error recovery by collaborating with other agents.

Chapters 4 and 5 describe the use of the TaMeX intelligent-agent framework. Chapter 4 illustrates the process of using the TaMeX intelligent-agent framework to develop an integrated application by using the book-finding example. It describes the development methodology, which is supported by a design-time toolkit, for creating and validating the various models needed, constructing the wrappers to allow communication with the web-based resources, and configuring the task agents needed to run the integrated application. Chapter 5 describes the running of the integrated application whose development was just described in Chapter 4. Similarly, to Chapter 4, it uses the book-finding example to illustrate the step-by-step run-time behavior of TaMeX.

The last two chapters conclude the thesis with an evaluation of TaMeX and a description of the contributions of TaMeX and concluding thoughts. Chapter 6 evaluates the current version of TaMeX by comparing it to the related works of the first version of TaMeX, other intelligent integration architectures and languages, and web services standardization work. As well, this chapter describes the prototypes developed using this version of TaMeX. Chapter 7 describes the contributions of this thesis and summarizes the future work.

Chapter 2 TaMeX Intelligent-Agent Framework

The integrated applications developed with the TaMeX framework share a common software architecture, defined in the framework. This chapter describes this intelligent-agent framework that is designed to address the following key issues critical to the development of an integrated web-based application:

- how to make the various parts of the application "talk" the same language,
- how to control the execution of the integrated application: i.e. how to order the steps, where to get the input from, how to evaluate whether progress is made, how to integrate intermediate results, and
- how to ensure that the user's specific needs and preferences are met.

To address these issues, several aspects must be developed: a common language, an execution-control mechanism and a user-customization facility. Table 1 shows these issues, along with the motivation for them and the TaMeX framework solution. The details of the TaMeX framework aspects that solve these issues are discussed in this chapter.

Table 1: Integrated Application Issues

Issue	**Motivation**	**TaMeX Framework Aspect**
Common Language	-so that various parts of the application can "speak" the same language	-domain model -wrappers
Execution Control	-to order the execution of steps -to know where & how to get input -to ensure correct step execution -to know how to integrate results	-task model -constraints -task agents
User-Customization Facility	-to accommodate the different preferences of different users	-profile model

The first section of this chapter describes a book-finding example that is used to illustrate the problems involved with manually accessing various web-based applications and manually combining and integrating this retrieved information. Then, throughout the thesis, this same example is used to illustrate the process of solving the problem by developing and then running an integrated application using TaMeX.

The second section describes the language that is used to specify the TaMeX models that represent several distinct yet related aspects of the integrated application. The aspects described are:

1. *Domain Model* - the "lingua franca" in which information is expressed and exchanged among the user's browser, the task agents and the wrapped applications;
2. *Task Model* - the integration workflow of the application, in terms of a hierarchical task structure;
3. *Constraints* - the relevant domain-specific semantic constraints, defining properties of the information exchanged and functional specifications of the workflow tasks; and
4. *Profile Model* - the end-user preferences.

The third section describes the multi-agent architecture of the TaMeX framework. It starts with a description of the architecture of an individual task agent. This TaMeX task agent interprets, at run-time, the executable specification of the TaMeX models. This section concludes with an example multi-agent architecture created by extending the TaMeX application framework.

The last section describes the book-finding application that was developed by applying the TaMeX framework to the comparative shopping example described in the first section of this chapter.

2.1 Book-finding: An Integrated-Application Example

Consider, for example, an instructor who needs to decide on a textbook for a new course that she is teaching. She wants to browse various on-line applications of bookstores. As a knowledgeable consumer, with specific constraints and preferences, she needs to access through her browser several different on-line application sites to identify the available options. And since she wants to compare the various options in terms of several dimensions, such as prices, publishers and authors for example, she will have to "manually" collect the retrieved candidates, transform them in a common format and compare them, prior to making a decision.

The instructor, as a consumer in this scenario, is an "information hunter" involved in finding information from multiple web sites and comparing this information to make a decision on which textbook to buy. This process of making comparisons is much more difficult than previously thought. Jared Spool found that it required significantly more energy for a consumer to answer comparative questions than the non-comparative questions. Consumers felt drained of their energy after answering a comparison question - their reported energy level was consistently lower than the lowest energy level reported after a non-comparison question [Spool99].

To help the instructor in keeping her energy levels up by assisting her decision-making, a book-finding application was developed in TaMeX to address this type of scenario. Before discussing the specifics of this application, the fundamental building blocks of the TaMeX intelligent-agent framework will be discussed in the following sections. Then the TaMeX book-finding integrated application will be described in Section 2.4.

2.2 The Integration-specification Language

The process of using the TaMeX framework to develop a multi-agent application for integrating a set of existing web applications relies on the specification of a related set of models. These models represent several distinct yet related aspects of the integrated application: the application domain, the task structure of the integrated application, the semantic constraints on the domain concepts and tasks, and the end-user customizations. Once these models have been specified, they constitute an executable specification that the TaMeX task agents can interpret at run-time.

The syntax of the TaMeX integration-specification language is based on XML [XML] and its related languages [XSD, XSL, XSLT]. These languages were chosen because of their extensible object-based syntax and of the immense tool support available for processing XML on the Web. The semantics of the TaMeX integration-specification language are inspired from the SBF-TMK [SG99] functional representation language.

The most important requirement for the design of an integration-specification language is extensibility, which is also a means for supporting expressiveness. This is why the TaMeX integration-specification language is based on a four-layer structure. Within this structure, each lower-level layer can be viewed as the implementation of the layer above it, with the uppermost layer being the most abstract and the lowest-level layer being the most concrete [Rap97]. Each layer implements the semantics of the syntax of the layer immediately above it. At the same time, the semantics of its own syntax is implemented by the layer immediately below it. As each successive layer is implemented, more meaning is added to the abstraction layer above it and the information model becomes more concrete, until the lowest-level layer is reached, which contains the specific data. This design enables the extensibility of the language: if new syntactic primitives are needed at one layer, then only the layer below it needs to be extended to implement the semantics of these new primitives. The layers of this architecture, along with their functions, are shown in Table 2.

The top-level layer provides the basic set of abstractions from which to construct models; these abstractions are essentially provided by the XML Schema Definition Language (XSD) and the XML Stylesheet Language (XSL). XSL includes an XML vocabulary for specifying formatting and XSL transformations (XSLT) for transforming XML documents into other XML documents.

At the next level of abstraction there are three XML meta-models, each one designed to describe a different language aspect: the domain model, the task model, and the profile model. The format of implementation for each layer is shown in brackets following the name of that particular layer. For example, the Task Model Schema is implemented using an XML Schema. These XML meta-models are described in detail in the following sections.

Table 2: Four Layered Integration-Specification Language - see Appendices A through C

Layer	Domain Model	Task Model	Profile Model
Meta-meta-model -infrastructure for a meta-modeling architecture -defines language for specifying meta-models	Basic Set of Abstractions: XML Schema Definition Language (XSD) XML Stylesheet Language (XSL)		
Meta-model -semantic interpretation of a meta-meta-model -defines language for specifying a model	Domain Model Template (XML Schema)	Task Model Schema (XML Schema)	Profile Part of Domain Model Template (XML Schema) + Profile Adaptation Stylesheets (XSL Stylesheets)
Model -semantic interpretation of a meta-model -language for specifying an information domain -application-specific layer	Domain Model Schema (XML Schema) (XSL Stylesheets)	Task Model Instance (XML Schema Instance) (XSL Stylesheets)	Profile Adaptation Model (profile-adapted Task Model Instance) + (profile-adapted Domain Model Schema)
User-Session Data -semantic interpretation of a model -defines a specific information domain	Run-time Domain Model Instance (User DOM)	Run-time Task Structure (User DOM)	Profile-adapted Run-time Task Structure (User DOM)

2.2.1 The Domain Model

The *Domain Model* of the TaMeX integration specification addresses the aspect of a common language for the integrated application. The Domain Model describes the concepts and the relations of the integrated application, and at the same time, establishes a target language for the existing web-application wrappers. All information exchanged at run-time between the integrated-application user, the task agents and the wrappers is represented in terms of the domain model. More specifically, the domain model specifies

- the basic entities of the application domain,
- their attributes,
- the composition relationships among them resulting in more complex entities, and
- the logic constraints on these entities that express domain-specific semantics.

Meta-model

The domain meta-model, called the *Domain Model Template*, is a generic specification that is applicable to defining the domain concepts for a family of related applications. In addition to these generic concepts, the template also contains concepts that must be adapted to the particular application. For these application-specific concepts, the names, attributes, and compositions must be filled in where designated within the *Domain Model Template*.

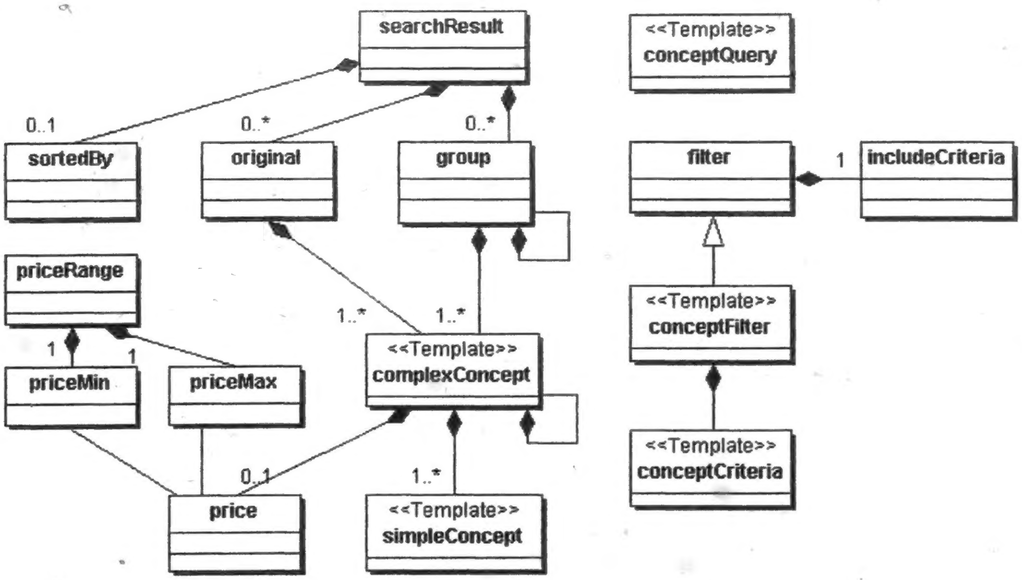

searchResult
<<Template>>
conceptQuery
0..1
0..*
0..*
sortedBy
original
group
filter
1
includeCriteria
priceRange
1..*
1..*
<<Template>>
conceptFilter
1
1
priceMin
priceMax
<<Template>>
complexConcept
<<Template>>
conceptCriteria
0..1
1..*
price
<<Template>>
simpleConcept

Figure 1: Domain Model Template UML Diagram for "comparative shopping"

The *Domain Model Template* is specified using an XML Schema. For example, the main part of the template for the "comparative shopping" application family is shown in Figure 1 using UML notation [UML]. The model of our example application, the book-finding application, was developed using this "comparative shopping" *Domain Model Template*. The concepts *price* and *filter*, seen in the UML diagram, are examples of the reusable generic concepts within the template. *Price* is the cost of the product or service being compared in the application. The associated generic concepts, *priceRange, priceMin*, and *priceMax* are used for filtering the goods/services returned by a user-specified price range. *Filter*, along with its associated concepts of *includeCriteria, conceptFilter*, and *conceptCriteria*, are the concepts used for the filtering of the returned goods/services according to user-specified criteria. Notice that *conceptFilter* and *conceptCriteria* are depicted in the diagram with the <<Template>> stereotype. This indicates that these concepts need to be changed during implementation of the model layer to reflect the application-specific names, attributes, and compositions. In other words, the concepts indicated by the <<Template>> stereotype are the application-specific concepts within the template.

The XML schema and its corresponding XML structure diagram for this Domain Model Template for comparative shopping are shown in Appendices A2 and A1 respectively. This XML schema and XML structure diagram include the profile model adaptations. For the UML representation of the profile model adaptations to the Domain Model Template for "comparative shopping" please refer to Figure 6 in Section 2.2.4.

Model

The model layer of the domain, called the *Domain Model Schema*, is the language for defining the information domain for a particular application within the family of related applications that is generically specified by the *Domain Model Template*.

The *Domain Model Schema* is specified with an XML schema that is based on the Domain Model Template of the application family. A UML representation of this schema for the book-finding application is shown in Figure 18. There are two types of extensions that may be necessary to produce a specific domain-model schema from a general domain-model template. First, new concepts, specific to the application, may be added. These new concepts may be basic or may be composed of other concepts, possibly already included in the domain-model template.

In the book-finding application, some of the new concepts added are *book, author*, and *publisherFilter*. *Author* is a basic concept, whereas *book* is composed of a domain-model template concept (*price*) and new application-specific concepts (*author, title, publisher, type, origin,* and *keywords*). Second, application-specific constraints may be added. The modeling of constraints is discussed in detail in Section 2.2.3.

The XML schema and its corresponding XML structure diagram for this Domain Model Schema for book finding are shown in the Appendices A4 and A3 respectively.

User-Session Data

The domain user-session data, called the *Run-time Domain Model Instance*, consists of the actual concept instances whose values are produced at run-time by the application end user, the task agent, and the wrappers. In our book-finding example, these values would include the criteria for the type of book searched for, the actual books found matching these criteria and their particulars. These values, and their associated type specifications, are stored in a Document Object Model (DOM) structure [DOM], maintained by the task agent interacting with the end user.

2.2.2 The Task Model

The *Task Model* in TaMeX specifies the workflow of the TaMeX integrated application. This workflow specification is used by the *task agent* to control the execution of the overall application including controlling the order of the execution of steps, the location and means of getting input, and the method of integrating results.

This information and control flow is specified in terms of a non-deterministic hierarchical task structure [SG96, SG97]. In this approach, a task is characterized by the type(s) of information it consumes as input and produces as output, and the nature of the transformation it performs between the two. A *complex task* may be decomposed into a partially ordered set of simpler subtasks. A simple, non-decomposable task, i.e., a *leaf task*, corresponds to an elementary procedure. The control of processing moves from higher-level complex tasks to their constituent subtasks. At the same time, information flows through the task structure as it is produced and consumed by the tasks.

The actual process is non-deterministic because there may be alternative decompositions for a given task, applicable under different conditions, and the subtasks resulting from the decomposition of a complex task are only partially ordered. Complex, high-level tasks get accomplished when all their required subtasks are accomplished.

Meta-model

TaMeX proposes a taxonomy of tasks, according to their operational semantics. There are two main types of tasks within this taxonomy: leaf tasks that correspond to the elementary services of the application and grouping tasks that define the composition of these services. The task structure meta-model, called the *Task Model Schema*, captures the semantics of this taxonomy and defines the language for specifying the task structure and control flow of an integrated application

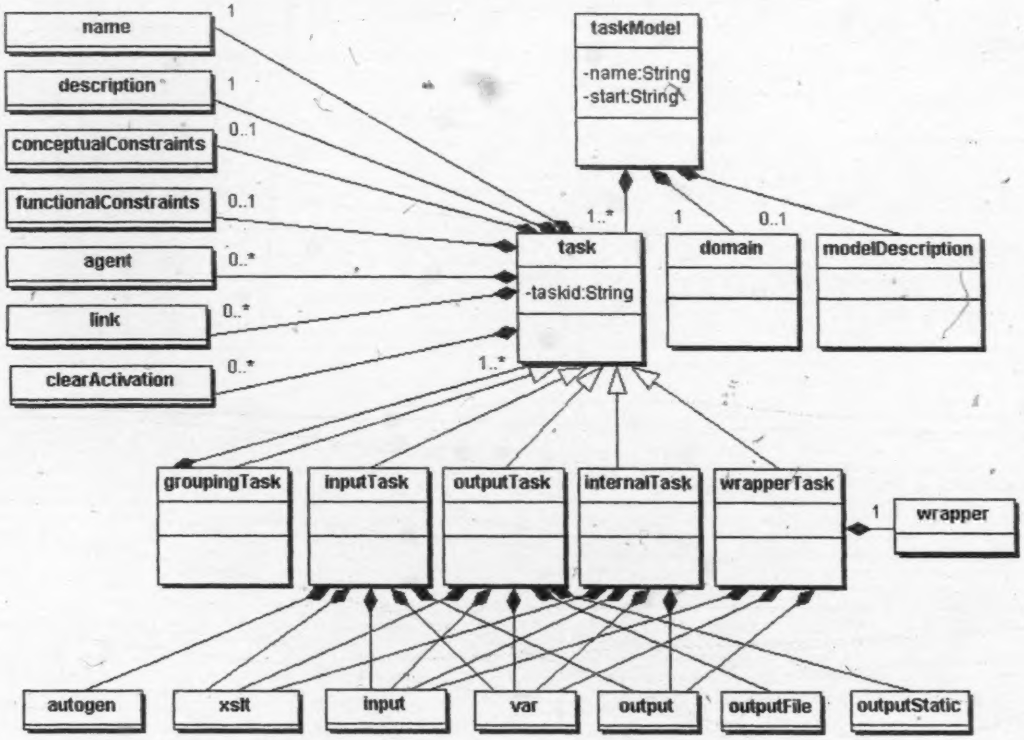

name
1
description
1
conceptualConstraints
0..1
functionalConstraints
0..1
agent
0..*
link
0..*
clearActivation
0..*
1..*
taskModel
-name:String
-start:String
1..*
1
0..1
task
-taskid:String
domain
modelDescription
groupingTask
inputTask
outputTask
internalTask
wrapperTask
1
wrapper
autogen
xslt
input
var
output
outputFile
outputStatic

Figure 2: Task Model Schema UML Diagram

The *Task Model Schema* is specified using an XML Schema. A pictorial representation of the Task Model Schema, in terms of a UML class diagram, is shown in Figure 2. From this diagram, it can be seen that a task model is composed of one domain, an optional model description, and multiple tasks. A task model has a *name* and a *start*ing task id for the first task in the hierarchy of tasks, A task can be either a *groupingTask*, which is composed of other tasks and specifies the control interdependencies of these subtasks, or a *leaf task*, which is an elementary procedure that consumes or produces information.

The leaf task types taxonomically specified are an *inputTask*, an *outputTask*, a *wrapperTask*, and an *internalTask*. *InputTasks* and *outputTasks* are *user-interaction* tasks. The execution of these types of tasks results in a new user-interface state in the end-user's browser. In the case of *inputTasks*, this is a new form, requesting user input and thus enabling the user to specify the task at hand. Correspondingly, *outputTasks* display information of interest to the user, effectively providing the "solution" to the problem whose parameters were specified with previous *inputTasks*. On the other hand, the execution of *wrapperTasks* results in accessing the wrapped web applications, using currently available information as input and contributing new data as output. *InternalTasks* are similar, in the sense that they also perform information transformations; however their execution results in invocation of new functional components, implemented for the purpose of the integrated application, usually in XSLT.

GroupingTasks specify the control interdependencies of their subtasks (either lower-level grouping tasks or leaf tasks). There are several types of inter-dependencies supported by grouping tasks: *sequencing*, when each task in the sequence depends on the previous one; *parallelism*, when a set of tasks are independent of each other but all have to be executed; *switch*, when a set of tasks are mutually exclusive; and *bag of tasks*, when only a subset of a set of independent tasks have to be executed.

Furthermore, a *groupingTask* may specify execution control over its subtasks - whether they are executed *automatically* as soon as they become possible or whether their execution is *user-controlled,* i.e., the user must invoke them through the integrated-application's interface.

Each task within this model has a unique *taskid* and is composed of task statements, some of which are common to all tasks and some of which are task-type specific. All tasks are composed of the task statements of a *name*, a *description*, an optional *conceptualConstraints* stylesheet, an optional *functionalConstraints* stylesheet, optional backup *agents*, optional *links* to other tasks, and optional *clearActivation* statements that set tasks to 'incomplete' when another task invalidates their completeness. The leaf tasks are composed of task-type specific data manipulating task statements such as *var*, *input*, and *output*. *Var* statements declare the variables to be used within the task, *input* statements load data from the user's DOM into the variables, and *output* statements take the variables and write their values back to the user's DOM.

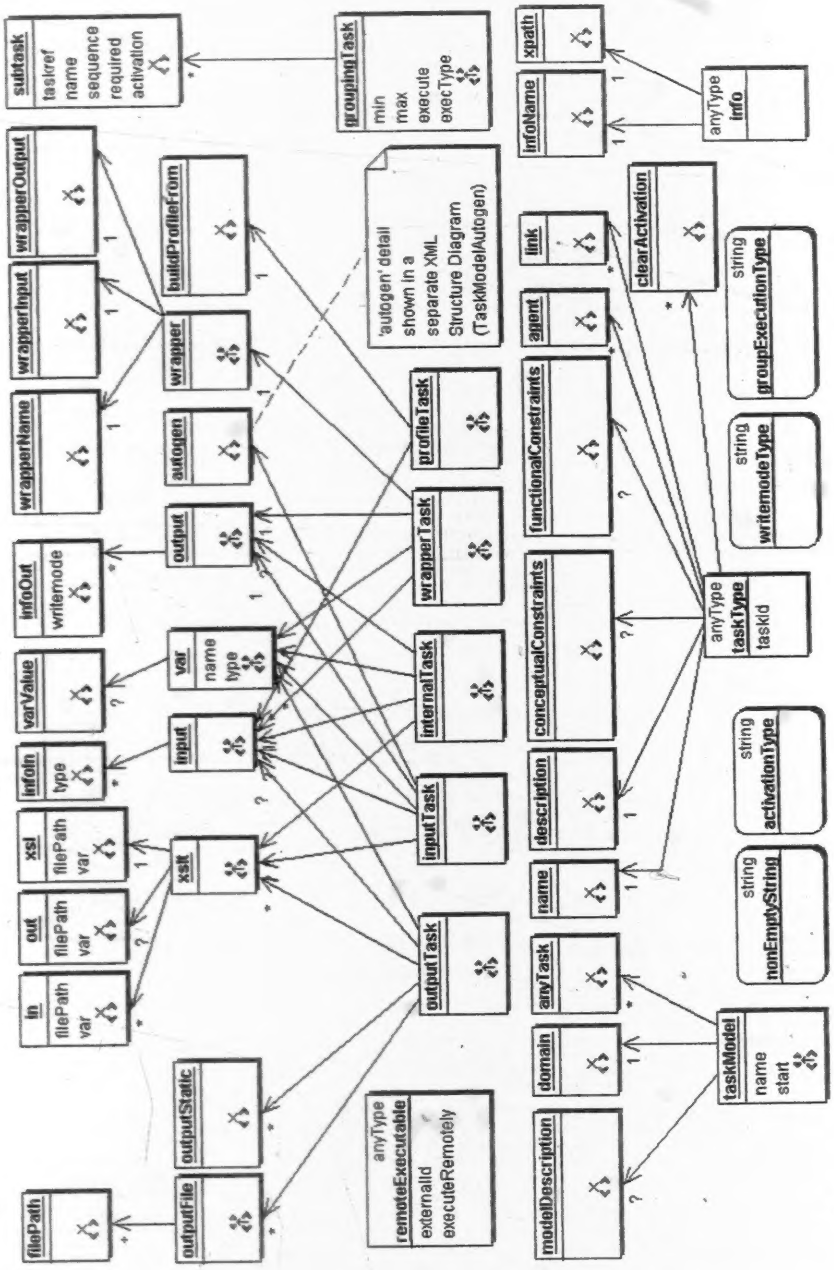

subtask
taskref
name
sequence
required
activation
groupingTask
min
max
execute
execType
xpath
infoName
anyType
info
wrapperOutput
wrapperInput
wrapperName
buildProfileFrom
wrapper
autogen
'autogen' detail shown in a separate XML Structure Diagram (TaskModelAutogen)
link
agent
clearActivation
string
groupExecutionType
profileTask
functionalConstraints
string
writemodeType
output
infoOut
writemode
wrapperTask
anyType
taskType
taskid
conceptualConstraints
var
name
type
varValue
internalTask
infoIn
type
input
string
activationType
xsl
filePath
var
xslt
inputTask
description
name
out
filePath
var
in
filePath
var
outputTask
anyTask
string
nonEmptyString
domain
taskModel
name
start
outputStatic
anyType
remoteExecutable
externalId
executeRemotely
modelDescription
filePath
outputFile

Figure 3: Task Model Schema - XML Structure Diagram

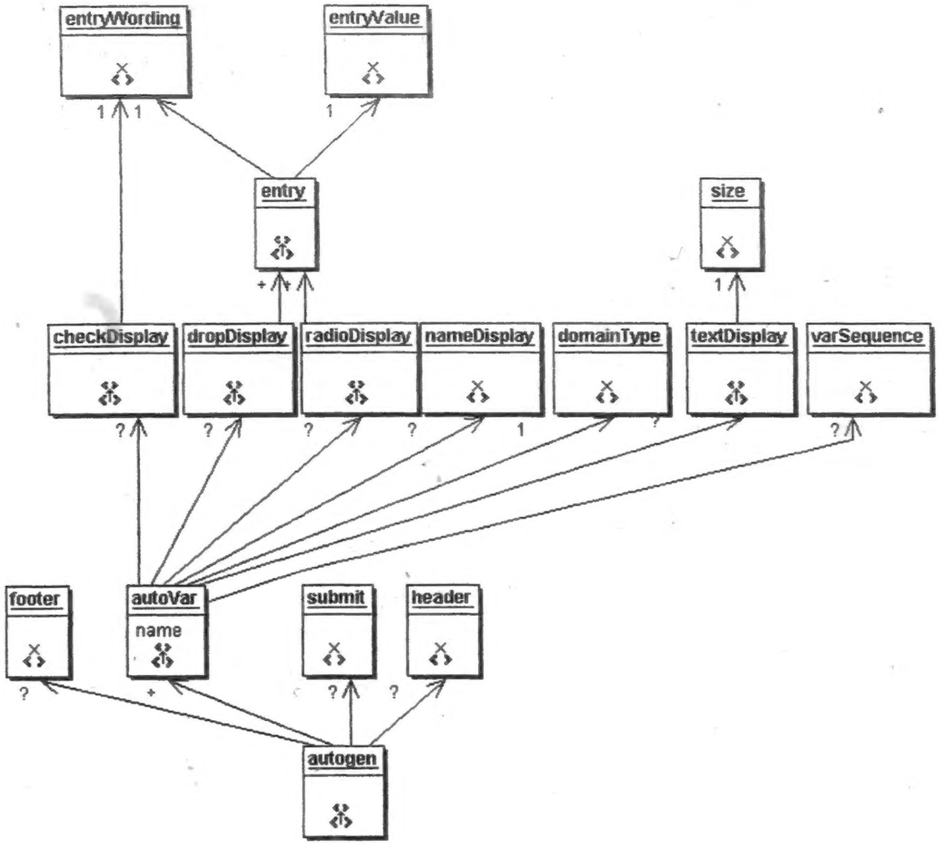

entryWording
entryValue
1
1
1
entry
+
+
size
1
checkDisplay
dropDisplay
radioDisplay
nameDisplay
domainType
textDisplay
varSequence
?
?
?
?
1
?
?
footer
autoVar
name
submit
header
?
+
?
?
autogen

Figure 4: "Autogen" Detail of the Task Model Schema - XML Structure Diagram

Task statements can also be composed of other elements. This level of detail is shown in the XML structure diagrams of Figures 3 and 4. Figure 3 represents the Task Model Schema and Figure 4 represents the detail of the *autogen* construct of the Task Model Schema. The *inputTask*, seen in the middle row of Figure 3, is the task used to query a user for input. Each task extends *taskType* to obtain the common task statements of *name, description, conceptualConstraints, functionalConstraints, agent*, link and *clearActivation* via inheritance. Additionally, *inputTask* is composed of the input task-specific task statements of *xslt, input, var, output*, and *autogen*. The xslt task statement, which describes the stylesheet application that will create the input form, is itself composed of the *in*, *out*, and *xsl* elements. The *in* element specifies where to obtain the input xml for the xslt application. The *out* element specifies where to put the output generated by the xslt application. The *xsl* element specifies where to obtain the xsl stylesheet for the application. The XML schema for the *Task Model* is shown in Appendix B1.

Model

The model layer of the task model, called the *Task Model Instance*, describes the tasks needed for a specific application, the composition of these tasks, the constraints on these tasks, and the presentation aspect of the user-interaction tasks involved. The Task Model Instance is specified using an XML schema instance of the meta-model and a set of application-specific XSL stylesheets.

The XML schema instance uses the language of the *Task Model Schema* to describe the actual tasks of a specific application and their composition. For example, in the book-finding application the tasks and their structure described in the XML schema instance are shown pictorially in Figure 5. In this diagram, the grouping tasks that are used to arrange the other tasks into sub-groupings are shown as regular solid boxes. The leaf tasks, which do the actual processing, are shown as heavy-lined solid boxes. The overall task of the book-finding application, i.e., to *find book information*, gets decomposed into the tasks *of specifying book selection criteria*, *accessing book-selling resources* via the wrapped web-based applications, arranging the accessed books by *sorting, grouping,* and/or *filtering,* and finally by *viewing the currently* selected and arranged books. There are two tasks to redo the book-finding process -- one to *clear the results* obtained and start over and the other to *reload the original* accessed results (i.e. before any arrangement of the results)

The application-specific stylesheets define the presentation aspect of the user-interaction tasks, implement the checking of the relevant constraints, and implement the functionalities of the internal tasks. For example, the stylesheet, shown in Appendix B3, implements a functional constraint for the *filter price* task within the book-finding application, i.e., that "the input minimum price is less than the input maximum price".

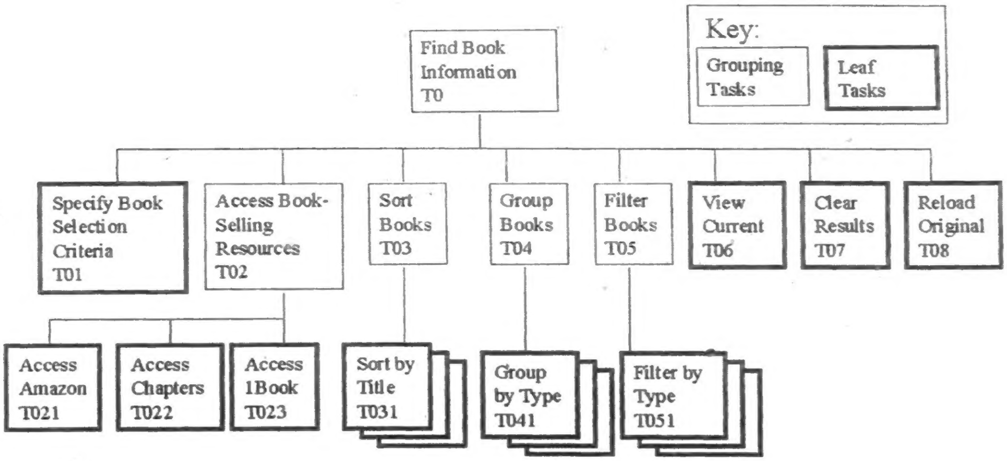

Key:
Grouping Tasks
Leaf Tasks
Find Book Information T0
Specify Book Selection Criteria T01
Access Book-Selling Resources T02
Sort Books T03
Group Books T04
Filter Books T05
View Current T06
Clear Results T07
Reload Original T08
Access Amazon T021
Access Chapters T022
Access 1Book T023
Sort by Title T031
Group by Type T041
Filter by Type T051

Figure 5: Task Structure for the Book-Finding Application

The XML schema instance for the book-finding application task model and a sample book-finding specific stylesheet that checks the constraint that "the minimum price is less than the maximum price" are shown in Appendices B2 and B3 respectively.

User-Session Data

The task user-session data, called the *Run-time Task Structure*, consists of values which are produced at run-time that indicate the current state of the end-user's problem-solving session. To accomplish this, the task agent is continually monitoring and updating the execution status of each task within the task structure. These values are stored in the task agent's DOM, maintaining the state of the user's problem-solving session.

2.2.3 Constraints

We have already mentioned the *constraints* aspect of TaMeX in the context of the domain and task models. The constraints define the domain-specific semantics of the application. At run time, the task agent evaluates whether the information values in its session state DOM conform to the constraints, to ensure that consistent progress is being made towards solving the user's problem.

As shown in Table 3, there are two main categories of constraints: *conceptual* and *functional*. Conceptual constraints are associated with the legal ranges of values and the composition of the domain concepts. An example of a conceptual constraint, shown in Appendix A5, is the check that *price*, if it has a value, must be numeric and may or may not have a '$' preceding the numeric price. Functional constraints are task-specific and characterize the nature of the information transformation expected of each task. The constraint of Appendix B3 is an example of a functional constraint, specifying a precondition on the filter by price task: namely that in order for the task to be accomplished, the values of its input parameter *priceMin* should be less than the value of *priceMax*. Constraint stylesheets are specified in terms of:

(a) the test to be performed in order to evaluate the constraint,
(b) the variables on which the test should be performed, and
(c) the error message to be generated if the test fails.

Table 3: Constraints used by Reflective Monitoring in our Framework

	Constraint Type	**Description**	**Where Specified**
Conceptual	Value Constraint Single Field	-constraints on the value of a single field of information in the domain model	Domain Model Schema XSL stylesheet
	Value Constraint Multiple Fields	-semantics on the relationship between values of multiple fields in the domain model	XSL stylesheet
	Compositional Constraint	-constraints on the composition of a concept	Domain Model Schema XSL stylesheet
Functional	Structural Constraint	-semantics on the relationship between the tasks in the task model	Task Model Schema and Instance
	Task Parameter	-constraints on the value of a single task parameter -semantics on the relationship between the values of attributes of the task parameters	Extension to Domain Model Schema XSL stylesheet

Constraints are specified using XML schemas and XSL stylesheets. Simple constraints involving the checking of the value of a single domain concept or task parameter are specified within the Domain Model Schema for the application, which is an XML schema. More complicated constraints involving constraints on the relationship between multiple concepts cannot be specified within an XML schema. These constraints are specified using XSL stylesheets. Structural constraints, which define the relationship between the tasks in the task model, are specified within the Task Model Schema and the Task Model Schema Instance for the application.

The constraints specified using XML schemas are shown in Appendices A4, B1, and B2 for the Domain Model Schema for the book-finding application, the Task Model Schema, and the Task Model Schema Instance for the book-finding application respectively. Examples of constraints specified using XSL stylesheets are the two examples just mentioned. The *price* conceptual constraint stylesheet and the *filter-by-price* task functional constraint stylesheet are shown in Appendices A5 and B3 respectively.

2.2.4 The Profile Model

The fourth aspect of the TaMeX language is the *profile model*. Its purpose is to enable the customization of the non-deterministic task-structure model according to individual user preferences. There are three types of customization that the end user of a TaMeX application can perform: task structure customization, execution control, and default value definition. These customization types are shown in Table 4.

Table 4: User Profile Customization Types

Type	**Description**
Task Structure Customization	-customize the task structure to suit the end user's needs by removing unwanted tasks
Execution Control	-modify the degree of control that the end user wishes to exercise on the execution of the task model. For example, s/he can specify whether a task should be executed automatically or left to be manually selected for execution by the user.
Default Value Definition	-define default values for various concepts manipulated by the task structure

Task structure customization is used by the end user to simplify the displayed task menu hierarchy by removing all the unwanted/unneeded tasks. Execution control customization is used by the end user to increase the amount of automatic execution of the task model, which correspondingly reduces the amount of manual interaction needed to solve the request. Default value definition is used by the end user to pre-define default values needed for the input task concepts. This pre-definition decreases the manual interaction needed by the user during the later running of the selected task model.

For example, the instructor of our book-finding example can choose to have the system automatically access the book information for inexpensive books from all three wrapped web-based applications instead of interactively controlling at run time which of these should be accessed. Her example customization of the book-finding application is shown in Table 5.. As well as searching for inexpensive books at all the web sites, she also wants the results displayed in price or type (hardcover or paperback) order.

Table 5: User Profile Customization of the Book-finding Application

Category	**End User Customization**
Task Structure Customization	-removed all the grouping tasks -removed all the sorts except 'Sort by Type' and 'Sort by Price'
Execution Control	-chose to automatically access the book information from all three wrapped web-based applications -chose to automatically filter the retrieved books by price
Default Value Definition	-defined default values for an inexpensive ($0 - $20) price range of books to be filtered by price

The instructor starts by simplifying the task structure by removing the grouping tasks and most of the sorting tasks that she never wants to use. Since she always wants to retrieve inexpensive books from all possible web sites, she chooses to automate the accessing of all three web-based applications and the filtering of the retrieved books by price. She chooses to enter default information for the *filter by price* task since she always wants the same price range of books to be selected. She enters default values of $0 - $20 for an inexpensive price range for the *filter by price* task

After this customization the system will display a simpler task hierarchy - excluding the grouping tasks and most of the sorting tasks - on the instructor's browser. As well as having a simpler task hierarchy with which to interact, the instructor also has fewer interactions to perform. So, instead of the instructor entering the book selection criteria, accessing each of the three web sites individually, filtering the retrieved books by price to obtain the inexpensive books, she would only need to enter the book selection criteria and the system would automatically do the rest. She could then choose to interact with TaMeX to display the unsorted (but already filtered) results in either price or type sequence.

Meta-model

The profile meta-model consists of the *Profile Part of the Domain Model Template* and the *Profile Adaptation Stylesheets*. The *Profile Part of the Domain Model Template* is a generic specification for defining the profile domain concepts. This specification may contain generic concepts applicable to a family of applications that can be used exactly as specified. It also contains concepts that must be adapted to the particular application. The *Profile Adaptation Stylesheets* are a set of XSL stylesheets used to provide functionality for the profile adaptations. There is a stylesheet used to generate the input form for the profile task. There are stylesheets used to provide the functionality for particular types of adaptation, specifically for automating and removing tasks.

The profile domain concepts are specified by adding the additional concepts needed for the profile to the *Domain Model Template*, which is specified using an XML Schema. The *Profile Adaptation Stylesheets* are specified using XSL stylesheets.

For example, the profile part of the Domain Model Template for "comparative shopping" applications is shown in Figure 6, adopting the UML notation. The concepts *profileOptionsContainer* and *defaultsContaine*r, seen in the UML diagram, are examples of the generic concepts within the template. *ProfileOptionsContainer* is the container, which holds all the profile options for a given application. The associated generic concept, *profileOption*, is the generalization for the application-specific profile options - in this case the concepts of automating a task: *autoTnn* and removing a task: *remTnn*. Notice that *autoTnn* and *remTnn* are depicted in the diagram with the <<Template>> stereotype. This indicates that these concepts need to be changed during implementation of the model layer to reflect the application-specific names, attributes, and compositions. In other words, the concepts indicated by the <<Template>> stereotype are the application-specific concepts within the template.

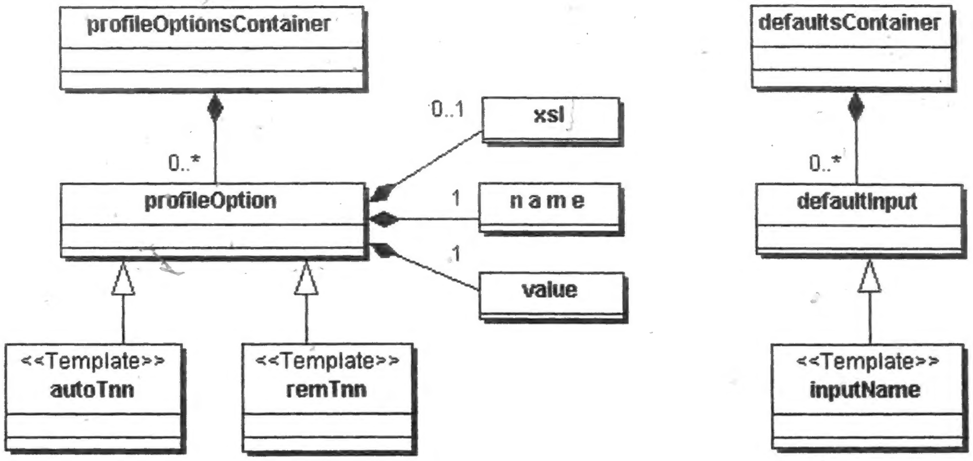

profileOptionsContainer
0..*
profileOption
0..1
xsl
1
name
1
value
<<Template>>
autoTnn
<<Template>>
remTnn
defaultsContainer
0..*
defaultInput
<<Template>>
inputName

Figure 6: Profile Part of Domain Model Template for "comparative shopping"

The XML schema and its corresponding XML structure diagram for the entire (including the profile part) Domain Model Template for comparative shopping are shown in Appendices A2 and A1 respectively. The *Profile Adaptation Stylesheets* are shown in Appendix C.

Model

The application-specific layer of the profile, called the *Profile Adaptation Model*, is the adaptation of the already created *Task Model Instance* and *Domain Model Schema* to define the profile adaptations that an individual user may perform within this particular application. These profile-adapted models are used at run-time to generate the input form for the profile task and correspondingly the profile options available for user-selection for each task.

The Task Model Instance is specified using an XML Schema Instance and the Domain Model Schema is specified using an XML Schema.

See Appendices A4 and B2 for an example of the profile-adapted Domain Model Schema and Task Model Instance from the book-finding application.

User-Session Data

The profile user-session data, called the *Profile-adapted Run-time Task Structure*, consists of values which are produced at run-time that indicate the current state of the end-user's profile-adapted problem-solving session. The run-time task structure, which now reflects the user's preferences, is used to generate the task hierarchy displayed on the user's browser, and control the application's execution. So the user sees and interacts with a task structure hierarchy that is individually configured to meet his/her needs. This is the result of the task-agent's profile-based customization capability. These values are stored in the task agent's DOM, maintaining the state of the user's profile-adapted problem-solving session.

2.3 The Architecture

Applications developed using the TaMeX framework are distributed multi-agent systems. A multi-agent system consists of a group of agents that interact and cooperate to accomplish some set of tasks in a distributed manner. Each agent offers a specific service or service composition to other agents. In TaMeX, the agents involved are called *task agents*. They control the processing between the users, the other task agents, and the web-based applications. The task agents are *intelligent software agents,* which is the new term used to describe agents situated in the environment of the Internet. A current working definition of *intelligent software agents*, given by Sycara et al. [SDPWZ96] is:

> "Intelligent Software Agents are programs that act on behalf of their human users in order to perform laborious information gathering tasks, such as locating and accessing information from various on-line information sources, resolving inconsistencies in the retrieved information, filtering away irrelevant or unwanted information, integrating information from heterogeneous information sources and adapting over time to their human users' information needs and the shape of the Infosphere."

2.3.1 The Individual Task Agent

In a TaMeX integrated application, the end-user browser contacts and interacts with a task agent containing a user interface agent. The contacted task agent, in turn, accesses the services of wrapped web applications through their corresponding wrappers and invokes other services that were originally developed to be used in the context of the integrated application. Furthermore, the task agent collaborates with the other task agents that contain a subcontractor agent by requesting and delivering services to the other agents in order to accomplish the overall task of the end user. A task agent – described in detail in Chapter 3 –uses the task model and its associated constraints to control the execution of the integrated application.

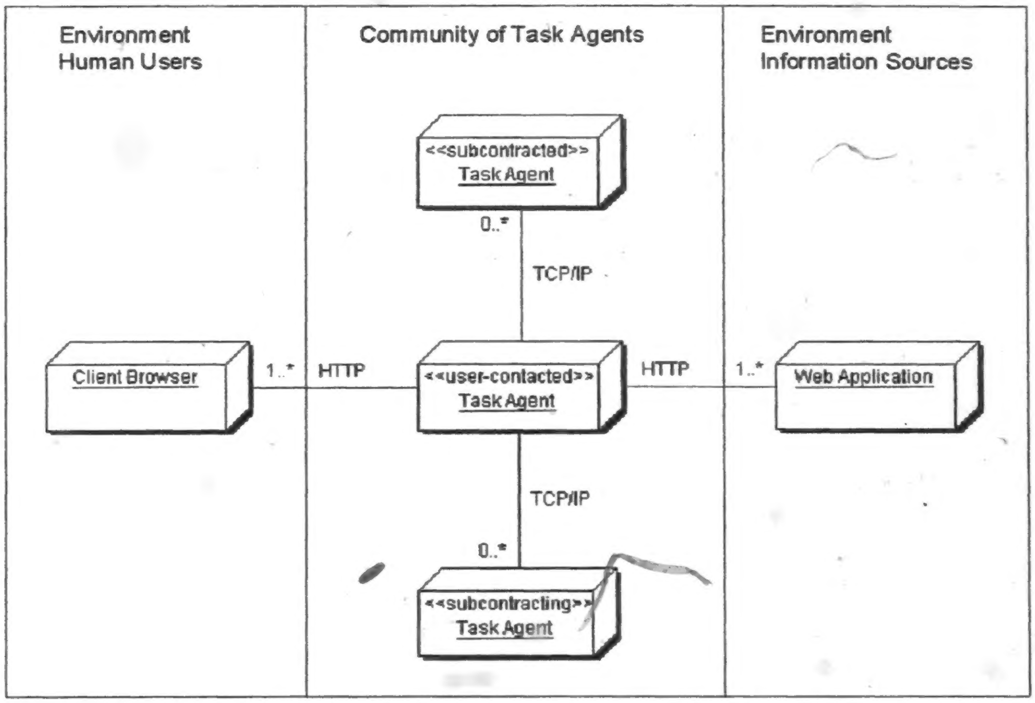

Environment
Human Users
Community of Task Agents
Environment
Information Sources
<<subcontracted>>
Task Agent
0..*
TCP/IP
Client Browser
1..*
HTTP
<<user-contacted>>
Task Agent
HTTP
1..*
Web Application
TCP/IP
0..*
<<subcontracting>>
Task Agent

Figure 7: The Architecture for a Task Agent

The architecture of the TaMeX application framework for a task agent is shown in Figure 7. The user-contacted task agent is in the center of the figure within the community of task agents. This task agent interacts with the other task agents in the community in two ways. Either it can subcontract out sub-tasks to the other task agents or it can be the recipient of sub-tasks that have been subcontracted to it by the other task agents. This task agent also interacts with the end user through a client browser and with the web applications that constitute the information resources for the integrated application.

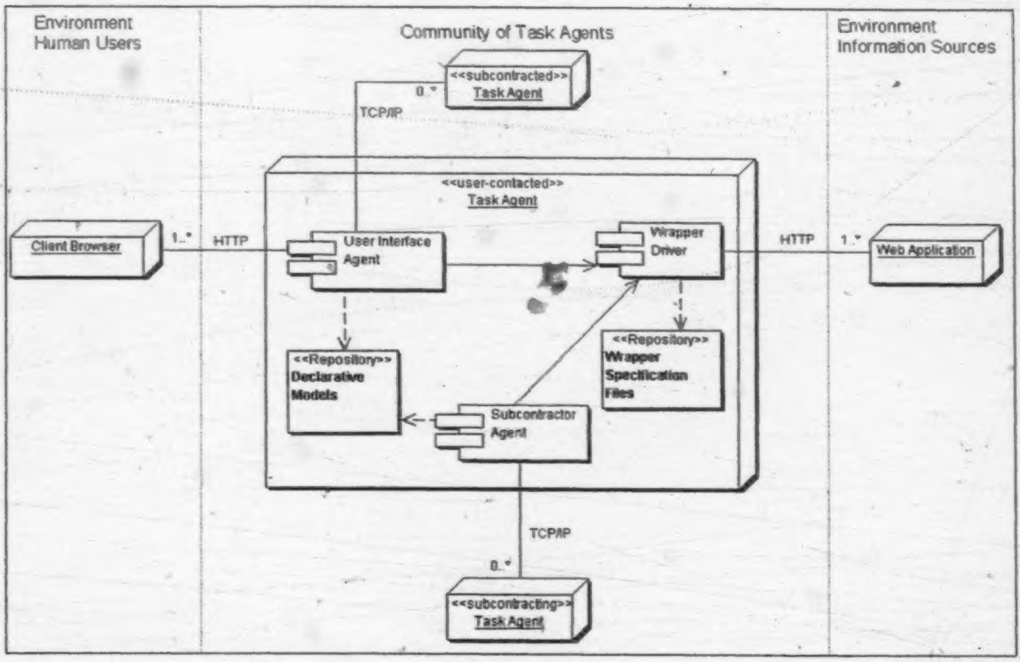

Environment
Human Users
Community of Task Agents
Environment
Information Sources
<<subcontracted>>
Task Agent
0..*
TCP/IP
<<user-contacted>>
Task Agent
Client Browser
1..*
HTTP
User Interface
Agent
Wrapper
Driver
HTTP
1..*
Web Application
<<Repository>>
Declarative
Models
<<Repository>>
Wrapper
Specification
Files
Subcontractor
Agent
TCP/IP
0..*
<<subcontracting>>
Task Agent

Figure 8: The Detailed Component Diagram for a Task Agent

Figure 8 depicts the expansion of the <<user-contacted>> task agent, henceforth referred to as the "orchestrating agent", in Figure 7 to show its detailed components. Each task agent consists of a repository of wrapper specification files, a wrapper driver component, a repository of declarative models used by this agent, and two interfaces for accessing this task agent.

The *repository of wrapper specification files (wrappers)* holds the specifications needed to allow the integrated application and the other web-based applications to communicate with each other. Wrappers encapsulate the existing web-based applications, whose services are being integrated. They provide the declarative representation of the transformations required to enable the access of these underlying web-based applications through a uniform mechanism. These wrappers are application-specific and are constructed, at design-time, for each web-based application to be accessed by the integrated application. TaMeX provides support to developers to construct wrappers semi-automatically using an example-based learning method [STS00]. See Section 4.4 for a description of the wrapper construction process.

The *wrapper driver component* uses these *wrapper specification files* to execute existing web applications, whose services are being integrated, and translate between the domain model of the integrated application and the individual domain models of the wrapped applications. More specifically, its role is to support the execution of the wrapper tasks of the task model. First, it translates the XML data provided as input to a wrapper task into an appropriate HTTP request to the underlying application, and second, it extracts the output expected of this task from the HTML responses of the application. For both these transformations, it uses the *wrapper specification files* to provide the translation specifications. The wrapper driver component is part of the TaMeX framework.

The *repository of declarative models* holds the specifications for the behavior and configuration of each task agent. The behavior specifications are represented in the TaMeX integration-specification language and were previously described in Section 2.2. These models define the concepts and relations of the domain model of the integrated application, the information and control flow among the various services (both the newly-developed and wrapper-invoked ones), and the semantic constraints on the domain concepts and the tasks. Parts of these behavior specifications are included in the TaMeX framework and other parts are application-specific and must be created by the developer at design-time. The meta-model layer of the TaMeX integration-specification language models is part of the TaMeX framework and the model layer must be created for each specific application. See Sections 4.1, 4.2, and 4.3 for a description of the domain model development, task model development, and profile model development respectively.

The configuration specifications in the *repository of declarative models* consist of two models for configuring the various aspects of the integrated application. The Main Configuration Model describes the numbers of the task agents, wrappers, and task model repositories of the application and their connectivity, as well as identifying the path to system components such as the stylesheets used for validation of the task models and formatting of the user interface display. The User Interface Model specifies the interface between the end user and the task agent. These models are described in the following sections. Part of the Main Configuration Model is included in the TaMeX framework and part is application-specific and must be created by the developer at design-time. The generic definition of the configuration options is part of the TaMeX framework. However, the developer must create the configuration for a particular task agent at design-time. The User Interface Model is part of the TaMeX framework. See Section 4.5 for a description of the task agent configuration.

The components for the *two interfaces for accessing* the task agent are a user interface agent and a subcontractor agent. The user interface agent provides services to the end users and the subcontractor agent provides services to other task agents - i.e. subcontracts its services. In other words, a task agent can be contacted by a user or by another agent. These components are part of the TaMeX framework. However, the configuring of a task agent to use one or both of these interfaces is done on an application-specific basis at design-time. See Section 4.5 for a description of task-agent configuration.

2.3.2 The Main Configuration Model

The *Main Configuration Model* is a declarative specification for configuring the system for a task agent. This model essentially describes the numbers of the task agents, wrappers, and task model repositories of the agents and their connectivity. The configuration categories and elements of this model are shown in Table 6 and its XML structure diagram is shown in Figure 9.

There are three registries specified in the Main Configuration Model: agent, task model, and wrappers. The agent registry, *registeredAgents*, contains information necessary to access each of the other agents that could be subcontracted by this task agent. The details of each agent entry in the agent registry are shown in Table 7.

Table 6: Main Configuration Model Categories and Elements

Configuration Category	**Element Name**	**Description**
Task Agent Identification	servletName	• specifies the full name of the Java servlet which implements the task agent
User Interface	staticIntro	• optional • specifies the path to the file of text that will appear at the top of the Task Model Selection Screen
	newUserIntro	• optional • specifies the path to the html code that will be displayed on the Task Model Interaction Screen when the user first selects a task model
	mainDisplayXSL	• specifies the path to the XSL stylesheet that formats the user interface display
Validation	strictValidationXSL	• specifies the path to the XSL stylesheet that will do the strict validation of the task models
Agents	agentPort	• specifies the port that this task agent will listen for requests on
	startAgent	• flag that is set to 'true' if the agent server for the subcontractor agent is to be started
	registeredAgents	• registry of access information for the agents that could be subcontracted by this task agent
Task Models	registeredTaskModels	• registry of information for the task models that this task agent is able to execute
Wrappers	registeredResources	• registry of information for the wrappers that this task agent is able to use to gather data

Table 7: Agent Registry (*registeredAgents*)

Element Name	**Description**
agentId	• specifies the unique id of this agent
agentName	• specifies the agent name (not used by the run time environment)
address	• address of the server that is hosting this agent being specified
port	• port number that this agent has been configured to listen for requests on

The task model registry, *registeredTaskModels*, contains information about the task models that this task agent is able to execute. The details of each task model entry in the task model registry are shown in Table 8.

Table 8: Task Model Registry (*registeredTaskModels*)

Element Name	**Description**
modelName	• specifies the name of this task model • must match the name supplied in the accompanying task model file
path	• specifies the path to the task model file that holds this task model

The wrapper registry, *registeredResources*, contains information about the wrappers that the task agent is able to use to gather data from the existing wrapped web applications. The details of each wrapper entry in the wrapper registry are shown in Table 9.

Table 9: Wrapper Registry (*registeredResources*)

Element Name	**Description**
resourceName	• specifies the unique wrapper name • used to look up the wrapper
requestTemplate	• specifies the path to the request protocol file • this file allows the task agent to communicate with the resource
extractionGrammar	• specifies the path to the extraction grammar file • this file extracts the pertinent data out of the resource's response and arranges it into a set of objects that is defined in the domain model
cookies	• flag that is set to 'yes' if the wrapper will send cookies to the resource

The Main Configuration Model is specified using an XML Schema for the generic definition of the structure and options for system configuration. The application-specific definition is specified using an XML Schema Instance that conforms to this schema.

The Main Configuration Model schema is shown in Appendix D1. See Appendix D2 for an example of a Main Configuration Instance that includes the book-finding application.

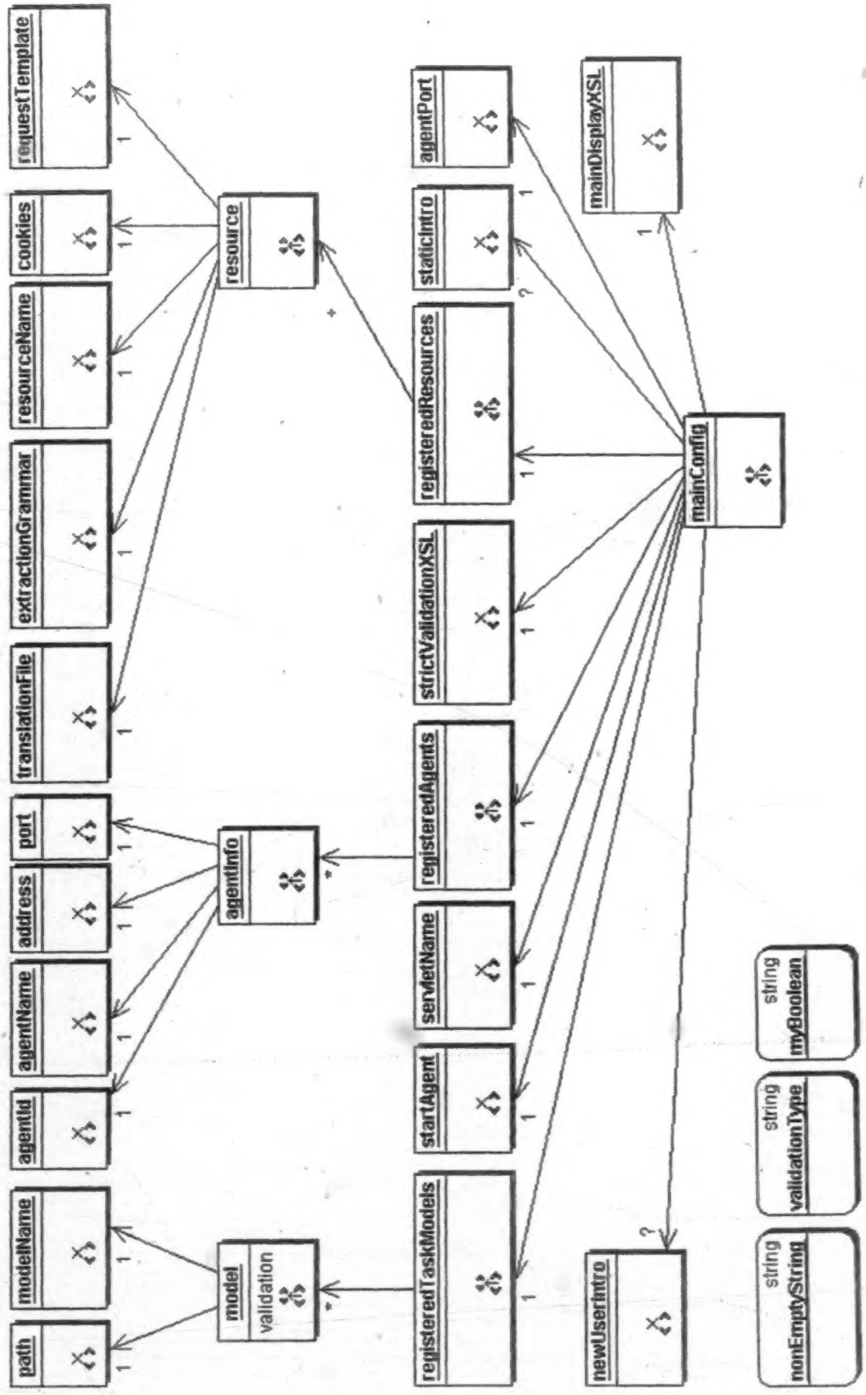

mainConfig
mainDisplayXSL
agentPort
staticIntro
registeredResources
strictValidationXSL
registeredAgents
servletName
startAgent
registeredTaskModels
newUserIntro
resource
requestTemplate
cookies
resourceName
extractionGrammar
translationFile
agentInfo
port
address
agentName
agentId
model
validation
modelName
path
string
myBoolean
string
validationType
string
nonEmptyString

Figure 9: Main Configuration - XML Structure Diagram

2.3.3 The User Interface Model

The User Interface Model specifies the interface that the task agent uses to interact with the end user. This model is part of the TaMeX framework and consists of the Main Display Schema and Stylesheet. The Main Display Schema defines the structure of the user interface XML that the task agent returns after an interaction with the end user. The Main Display Stylesheet uses this XML information to generate the user interface HTML that is displayed on the user's browser.

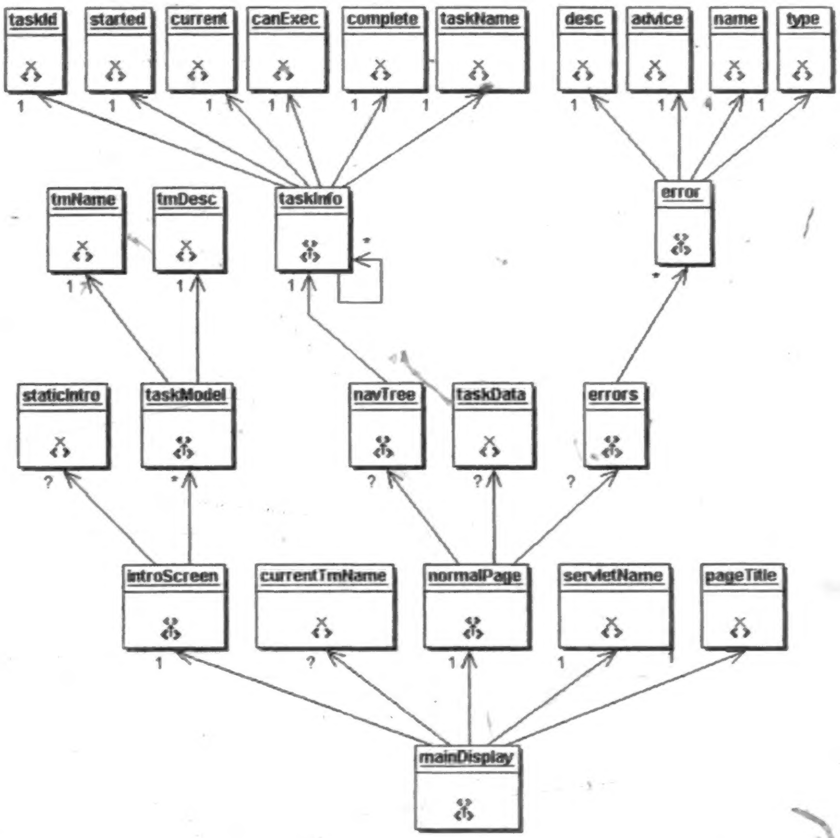

taskId
started
current
canExec
complete
taskName
desc
advice
name
type
tmName
tmDesc
taskInfo
error
staticIntro
taskModel
navTree
taskData
errors
introScreen
currentTmName
normalPage
servletName
pageTitle
mainDisplay

Figure 10: Main Display Schema - XML Structure Diagram

The Main Display Schema, as shown in the XML structure diagram of Figure 10, defines the structure for two types of user interaction screens: the task model selection display and the task model interaction display. TaMeX automatically generates the user interaction screens from the information specified by the developer in the declarative models and the information representing the current state of the user request processing. In other words, the developer does not need to explicitly enter the information needed for the user interaction.

For example, the *introScreen* subtree of the Main Display Schema is used to generate the task model selection screen, which displays a list of task models and their descriptions. The *taskModel* information of this subtree is obtained from the task model names of the *registeredTaskModels* element of the Main Configuration instance and from the task model descriptions of the *modelDescription* element of the corresponding Task Model Instance. The *normalPage* subtree of the Main Display Schema is used to generate the task model interaction screen, which displays either as a normal interaction page or an error page. The *navTree* subtree, i.e. navigational tree, of *normalPage* is used to display a hierarchical menu of tasks on the left-hand side of the task model interaction screen. The *taskData* element or the *error* subtree of *normalPage* is used to display normal interaction data or error data respectively on the right-hand side of the task model interaction screen.

The Main Display Schema and the Main Display Stylesheet file are shown in Appendices E1 and E2 respectively.

2.4 Book-finding Multi-Agent Application

An example multi-agent application created by extending the TaMeX framework to produce three task agents in the book-finding domain is diagrammatically depicted in Figure 11. This is the architecture for the book-finding example described in Section 2.1. It integrates three existing book-selling web applications and implements six sorting, four grouping, and six filtering services to support tasks such as comparative shopping. To assist the instructor in her decision-making and integrate the existing web applications along with the new services provided by the TaMeX application, three wrappers and three task agents were developed. However, it is important to note that TaMeX does not place any limit on the number of task agents and wrappers included in an application. Each wrapper is the book-finding service of one of the book-selling web sites adapted to TaMeX services - so three wrappers were developed corresponding to the book-finding service from each of the three book-selling web applications to be integrated. Each task agent is a different composition implementation of the TaMeX Book-finding application, implementing a subset of the available services. Therefore, each of the three task agents has different capabilities, in that its specifications enable it to access different wrappers and different new services such as a sorting or grouping facility. However, each task agent has the potential to access the full range of available services by collaborating with the other agents. This allows the end user to interact with the task agent that she knows about and still have access to all the available services. This collaborative process is transparent to the end user; she may accomplish the same tasks, irrespective of which task agent she may contact, because the contacted agent is responsible for working with other agents towards accomplishing the overall desired task.

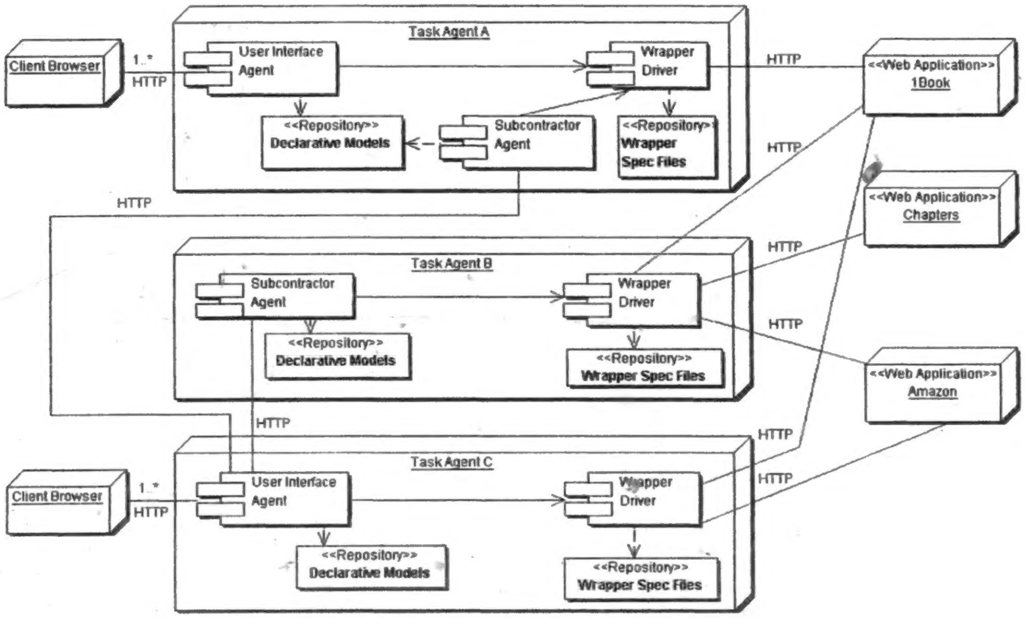

Client Browser
1..*
HTTP
Task Agent A
User Interface Agent
Wrapper Driver
<<Repository>> Declarative Models
Subcontractor Agent
<<Repository>> Wrapper Spec Files
HTTP
<<Web Application>> 1Book
HTTP
HTTP
<<Web Application>> Chapters
HTTP
Task Agent B
Subcontractor Agent
Wrapper Driver
<<Repository>> Declarative Models
<<Repository>> Wrapper Spec Files
HTTP
<<Web Application>> Amazon
HTTP
HTTP
Task Agent C
Client Browser
1..*
HTTP
User Interface Agent
Wrapper Driver
<<Repository>> Declarative Models
<<Repository>> Wrapper Spec Files
HTTP

Figure 11: The Component Architecture of our Framework for Book-Finding.

The three TaMeX task agents are shown in the middle of the diagram. Notice that *Task Agent A* contains both a *user interface agent* and a *subcontractor agent* and therefore can function both as a task agent for end users and for other task agents. *Task Agent B*, on the other hand, contains only a *subcontractor agent* and therefore can function only as a task agent for other task agents. *Task Agent C*, similarly to *Task Agent B*, has only one type of client - in this case end users. The browsers for the users that interact with these task agents are shown on the left side and the three web-based applications that interact with them are shown on the right.

Consider a scenario where the instructor contacts *Task Agent C*. *Task Agents A* and *B* have some task capabilities that *Task Agent C* lacks. For example, *Task Agent B* is configured to access the book-finding service of "Chapters" and *Task Agent A* can perform the task *Group by Type*, both of which are capabilities that *Task Agent C* does not have. *Task Agent C* will accomplish the instructor's overall task by traversing the book-finding task structure and performing the tasks within that structure. *Task Agent C* will either perform these tasks directly by itself, if capable, or will identify and then collaborate with other agents that are capable of performing these tasks. Also if *Task Agent C* is capable of performing a task but encounters an execution problem it will attempt error recovery by collaborating with other agents that are capable of executing this particular task.

Chapter 3 The Task Agent

After all four aspects - the domain model, the task model, the constraints associated with these models, and the profile model - of the integrated application have been represented in the TaMeX language, they become part of the task-agents' repositories (see Figure 2) thus defining their behavior. This chapter discusses the various functionalities of the TaMeX task agent, and the role that the relevant models play in supporting them. The main function of the TaMeX task agent is to control the flow of information by interpreting the task model. For this interpretation, the task agent uses either the standard task model as provided by the integrated application or the customizations defined in the user's profile model. This task model interpretation and user-customization of the task model are described in sections 3.1 and 3.2.

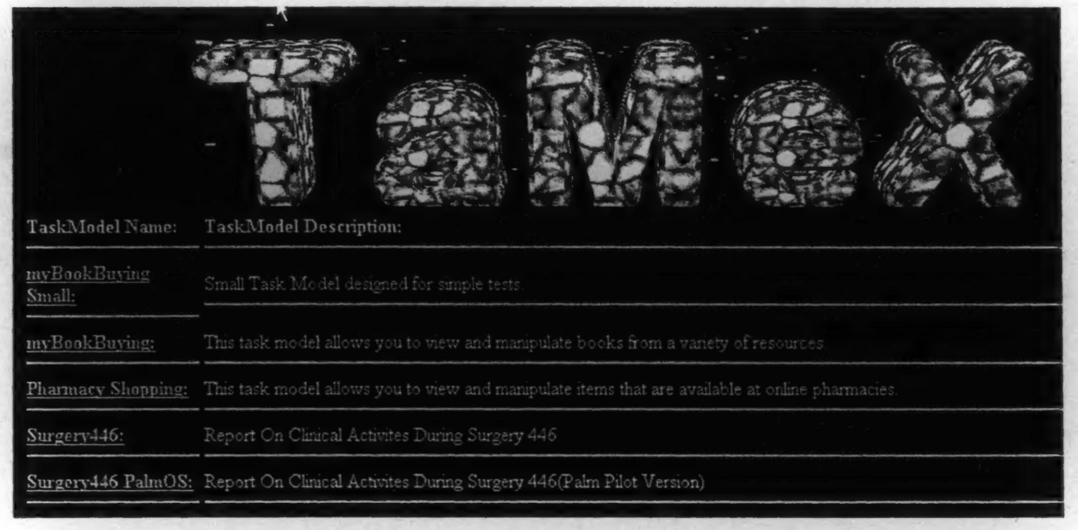
The TaMeX task agent also provides user interaction support by generating and displaying on the user's browser an initial screen for task model selection, followed by a task model interaction screen for the selected integrated application. The task model interaction screen, consists of a hierarchical task menu and an area for information input and display and shows the user exactly where s/he is in the execution of the application. The task agent is able to show the user this current state of affairs by maintaining the overall state of the task-model interpretation for this user's session. The user interaction support and the maintenance of the user-session state are described in sections 3.3 and 3.4 respectively.

The final functionality of the TaMeX task agent, described in section 3.5, is the reflective monitoring and recovery process. Throughout the run-time, the task agent checks the validity of information used and produced at each stage of execution. Whenever invalid information is encountered, it recognizes this as a failure and attempts error recovery. One of the ways that a task agent attempts to recover from a failure is by collaborating with other agents that have the capability to execute the task that failed.

3.1 Configurable Task-Model Interpretation

The task agent in TaMeX is a configurable component, implemented as a Java servlet. Its primary functionality is to interpret the task models in its repository. The task models are non-deterministic and the order in which the various tasks are executed at run time depends on the evaluation of the related constraints and the control exercised by the user at run time. This section gives an overview of the task-model interpretation process. The details of this process are described in Chapter 5.

3.1.1 Task Model Selection



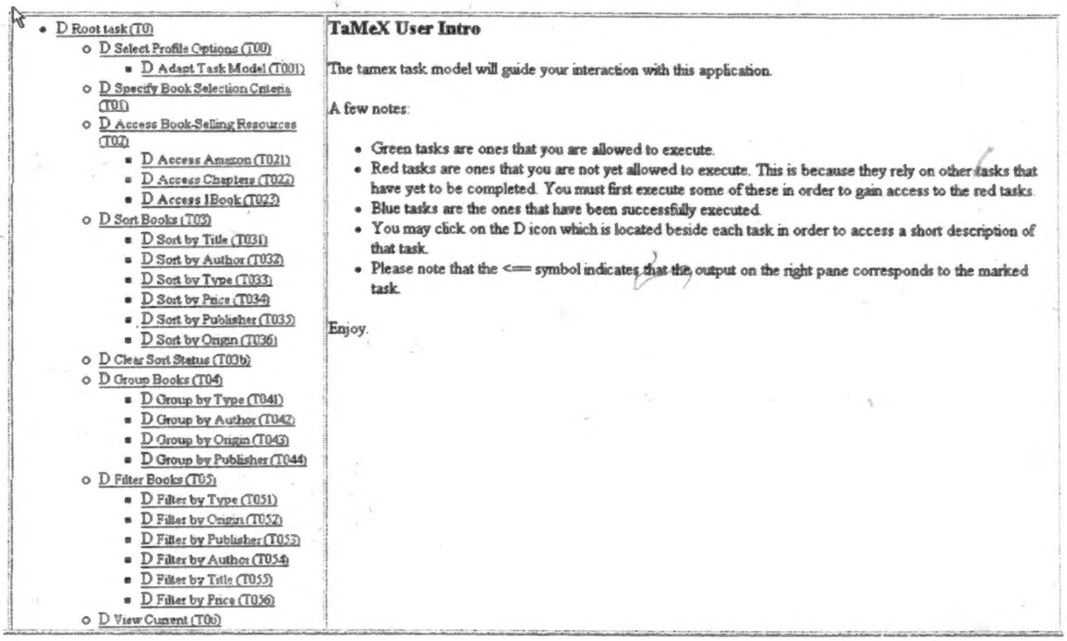
TaskModel Name:	TaskModel Description:
myBookBuying Small:	Small Task Model designed for simple tests
myBookBuying:	This task model allows you to view and manipulate books from a variety of resources
Pharmacy Shopping:	This task model allows you to view and manipulate items that are available at online pharmacies.
Surgery446:	Report On Clinical Activites During Surgery 446
Surgery446 PalmOS:	Report On Clinical Activites During Surgery 446(Palm Pilot Version)

Figure 12: Task Model Selection Screen

The task agent controls the communication with the application user via two types of user interaction screens: the initial *task model selection* screen and the *task model interaction* screen. The task agent returns the *task model selection* screen, as shown in Figure 12, when it is initially contacted by a new user. This screen displays the list of task models that are supported by this task agent.

3.1.2 Task Model Interaction

As soon as the application user chooses an integrated application to be run via this *task model selection* screen, the task agent starts executing the user task by interpreting the standard task model for the particular integrated application chosen. The interpretation process involves first identifying which task can be executed at each point in time. At each step, the subtasks or the sibling tasks of the current task become executable – the task model is traversed in a pre-ordered fashion – depending on the control interdependencies defined by the grouping tasks. Once an executable leaf task has been selected, then the task agent invokes the operation it implies, i.e., user interaction, wrapper invocation or internal functionality invocation.



- D Root task (T0)
  - D Select Profile Options (T00)
    - D Adapt Task Model (T001)
  - D Specify Book Selection Criteria (T01)
  - D Access Book-Selling Resources (T02)
    - D Access Amazon (T021)
    - D Access Chapters (T022)
    - D Access IBook (T023)
  - D Sort Books (T03)
    - D Sort by Title (T031)
    - D Sort by Author (T032)
    - D Sort by Type (T033)
    - D Sort by Price (T034)
    - D Sort by Publisher (T035)
    - D Sort by Origin (T036)
  - D Clear Sort Status (T03b)
  - D Group Books (T04)
    - D Group by Type (T041)
    - D Group by Author (T042)
    - D Group by Origin (T043)
    - D Group by Publisher (T044)
  - D Filter Books (T05)
    - D Filter by Type (T051)
    - D Filter by Origin (T052)
    - D Filter by Publisher (T053)
    - D Filter by Author (T054)
    - D Filter by Title (T055)
    - D Filter by Price (T056)
  - D View Current (T06)

TaMeX User Intro

The tamex task model will guide your interaction with this application.

A few notes:

- Green tasks are ones that you are allowed to execute.
- Red tasks are ones that you are not yet allowed to execute. This is because they rely on other tasks that have yet to be completed. You must first execute some of these in order to gain access to the red tasks.
- Blue tasks are the ones that have been successfully executed.
- You may click on the D icon which is located beside each task in order to access a short description of that task.
- Please note that the <== symbol indicates that the output on the right pane corresponds to the marked task.

Enjoy.

Figure 13: Task Model Interaction Screen

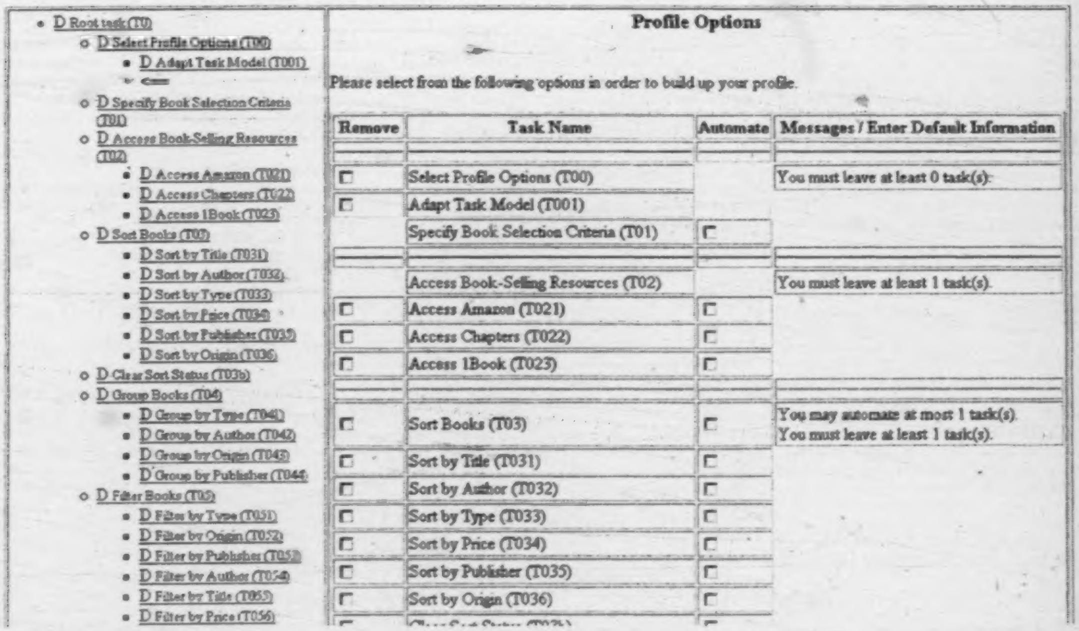
Once the task agent has started executing a task model, it uses the *task model interaction* screen, shown in Figure 13, to interact with the application user. This display consists of two sections: a *navigation area* on the left and a *user data area* on the right. To enable the user to control the interpretation of the task model, the task agent generates a hierarchical menu reflecting the hierarchical task model that is being interpreted. An example of this hierarchical menu is shown in the navigation area (left window) of Figure 13. The leaf tasks of the task model correspond to the most “indented” menu options; all other options correspond to grouping tasks. In addition, the current state of the interpretation process is communicated through the color of the various menu options: potentially executable tasks are shown in green, already complete tasks are shown in blue, and not yet executable tasks are red. The “red” tasks are not currently executable because they rely on other tasks that have not yet been completed. The user navigates this task menu by clicking on the “green” task of his/her choice. The arrow in this window points to the task that was last executed. The navigation area also provides some always-available options, such as accessing the task model selection screen, restarting the task model currently being executed, and accessing the "help" and "about" pages.

In addition to enabling the end user to control the execution of the integration workflow through the navigation menu, the task agent is also responsible for enabling the information input and display. The *user data area* of the *task model interaction* screen, shown on the right side of Figure 13, is used for this task-specific user interaction which includes input forms, output tables, applets, and recoverable errors. When, during the task-model interpretation process, a user-interaction task is selected, the task agent generates the user interface necessary for the interaction, in the right panel of the window. It does so by applying the XSL stylesheet associated with the selected user-interaction task to the current state of the process as contained in the agent’s DOM. For every output task there is an associated XSL stylesheet which is applied to the value of the "results" domain concept, as it is stored in the task agent's DOM to display the output. For every input task there is an associated XSL stylesheet or a Task Model Schema 'autogen' construct specified in the Task Model Instance which defines which information the user input will provide. Finally, errors in the process, discovered as violations of task- or domain-model constraints, are also reported in this panel.

3.1.3 Behavior Specialization

All task agents in a TaMeX application share some subset of the same task models, representing the service integration implemented in the application. However, even if they shared the exact same task models, the behavior of each individual task agent is specialized in two ways. First, each task agent is provided with the necessary details, specific to the access protocol, of the wrappers that it can invoke. As shown in Figure 2, these wrapper specifications are saved in the repository of the wrapper driver component for this task agent. So although some task agents could share the same task model which specifies the same list of wrapper tasks, each agent is able to access a different subset of them, depending on whether or not they have been configured with the access-protocol information. The wrapper configuration is done when the task agent is initiated and is described in Section 4.5. Second, the behavior of a task agent may be customized when contacted by an end-user's browser according to the user's profile. This user customization is done when the task agent is contacted by a user at run-time and is described in the following section and in more detail in Section 5.7.

3.2 Profile-based Customization



- D Root task (T0)
  - D Select Profile Options (T00)
    - D Adapt Task Model (T001)
  - D Specify Book Selection Criteria (T01)
  - D Access Book-Selling Resources (T02)
    - D Access Amazon (T021)
    - D Access Chapters (T022)
    - D Access 1Book (T023)
  - D Sort Books (T03)
    - D Sort by Title (T031)
    - D Sort by Author (T032)
    - D Sort by Type (T033)
    - D Sort by Price (T034)
    - D Sort by Publisher (T035)
    - D Sort by Origin (T036)
  - D Clear Sort Status (T03b)
  - D Group Books (T04)
    - D Group by Type (T041)
    - D Group by Author (T042)
    - D Group by Origin (T043)
    - D Group by Publisher (T044)
  - D Filter Books (T05)
    - D Filter by Type (T051)
    - D Filter by Origin (T052)
    - D Filter by Publisher (T053)
    - D Filter by Author (T054)
    - D Filter by Title (T055)
    - D Filter by Price (T056)

Profile Options

Please select from the following options in order to build up your profile.

Remove	Task Name	Automate	Messages / Enter Default Information
☐	Select Profile Options (T00)		You must leave at least 0 task(s).
☐	Adapt Task Model (T001)		
	Specify Book Selection Criteria (T01)	☐	
	Access Book-Selling Resources (T02)		You must leave at least 1 task(s).
☐	Access Amazon (T021)	☐	
☐	Access Chapters (T022)	☐	
☐	Access 1Book (T023)	☐	
☐	Sort Books (T03)	☐	You may automate at most 1 task(s). You must leave at least 1 task(s).
☐	Sort by Title (T031)	☐	
☐	Sort by Author (T032)	☐	
☐	Sort by Type (T033)	☐	
☐	Sort by Price (T034)	☐	
☐	Sort by Publisher (T035)	☐	
☐	Sort by Origin (T036)	☐	

Figure 14: Adapt Task Model Screen

The user can exercise control over the run-time behavior of the task agent by customizing the application. She may choose to use the standard hierarchical menu of tasks provided by the task agent 'as is' or use the profile model to adapt and automate portions of this task structure to suit her own individual preferences. The *Adapt Task Model* screen, shown in Figure 14, is used at run-time to enter her choices. The user performs the customization by selecting the profile options desired by checking the appropriate boxes, along with entering any needed default information. These user-specific customizations may involve changes to the degree that task execution is automatic or user-controlled, unneeded tasks are removed, and default inputs for some of the tasks are pre-specified. See Section 5.7 for a detailed example of this user profile customization process.

If the user customizes the application, the TaMeX task agent applies this user profile information to adapt the task model to reflect these customizations. Then, the task agent employs this newly-configured task model to determine its behavior instead of the standard task model. It continues its processing using the user-customized task model as its driver. The process of traversing the task model is the same as described in the previous section. However, the user-customized task model now drives the behavior of the task agent and therefore is displayed to the user in the hierarchical menu of tasks. The user would then continue the process of navigation using this new menu.

3.3 User-Session State Maintenance

The task agent maintains the overall state of the task-model interpretation, i.e. the session, of each user interacting with this agent in a Document Object Model (DOM). There is one DOM saved for each user and this DOM exists for the entire time that the user's session is active. When a particular user's session is considered finished - i.e. when there has been no interaction for a specified time period (currently two hours) - the task agent removes the DOM for this user. The DOM contains the values of all types of information produced during processing, i.e., data input by the user and generated by internal and wrapper tasks, and a record of which tasks of the task model have been executed and in what order. The DOM is used as a central repository of the user-session information. The input parameters required by each executed task are retrieved from it, and the output produced by the task execution is stored back to it. Thus, the DOM continually grows with information as the user progresses through a task model interaction.

3.4 Reflective Monitoring, Failure Detection and Inter-agent Collaboration

In addition to simply executing the task model, the task agent also monitors the progress of the process, recognizes failures as violations of the specified constraints and collaborates with other agents in the application to recover from these failures. The task agent monitors the validity of information used and produced at each stage of execution by checking the constraints specified in the models. First, the task agent interprets all information as it is produced according to domain-specific conceptual constraints. Second, the task agent interprets all input information needed for execution of a task according to the task parameter functional constraints. Third, when a task is completed, the task agent evaluates the functional constraints applicable to the task to verify that the task semantics are met.

When any of the specified constraints is violated, the task agent recognizes a failure in the task-model interpretation process, and attempts to recover. Depending on the problem, different actions may be taken. For example, if the problem is invalid data from the user, such as the input of a wrong price range, the task agent will return an error message. Furthermore, it will consider the current task as not yet completed, and it will continue to interact with the user until valid data is returned.

If the problem is that a wrapper task cannot be executed, because the wrapper accessible to the task agent does not have the protocol-grammar specification of the underlying web application, the task agent will collaborate with other agents to attempt error recovery. It will identify another agent that has access to an appropriately equipped wrapper and it will subcontract it to execute the required task. The "manager" task agent will send a message to the identified "subcontracted" agent with the information needed to execute the task-in-question. If the manager task agent receives appropriate results from this sub-contracted agent, it continues with its processing, otherwise it sub-contracts the task to another agent on the list.

Chapter 4 Application Development with TaMeX

The process of developing an integrated application using the TaMeX framework is described in this chapter. This development methodology is supported through a design-time development toolkit, which consists of

- templates and schemas for creating declarative models in the integration-specification language for a specific application,
- a tool for semi-automatically constructing wrappers using an example-based learning process, and
- a set of validating tools for the various required specifications that are represented in XML-based syntax.

The TaMeX development process involves developing the application-specific layer of the integration-specification language, which is the model layer. The model layer implements the generic meta-model layer above it, which was developed as part of the TaMeX framework and is used as a base for developing new integrated applications.

This process of creating an application using the TaMeX framework is illustrated by a step-by-step procedure using the book-finding example described in Chapter 2.1. Since some steps in creating new applications are common for different multi-agent frameworks, TaMeX's process incorporates some steps from other frameworks. The list of TaMeX steps combines steps proposed by Nodine, Perry, and Unruh [NPU98] for the InfoSleuth framework, steps proposed by Brugali and Sycara [BS98] for the RETSINA framework, with steps needed to implement the TaMeX integration-specification language.

The preliminary steps needed to create a new information integration application using the TaMeX framework are as follows.

1. First, the concepts of the particular application area must be identified. This domain ontology, which is a structured vocabulary, provides a basis for sharing semantics between the task agents and the components of the system.

2. Second, the family of related applications to which this particular application belongs must be identified. For example, the book-finding application belongs to the family of "comparative shopping" applications.
3. Finally, the key abstractions in the problem area being studied must be identified.
    1. *Sources of Information* - Find the web sites providing the services needed for this application. For each web site found, identify a corresponding wrapper task
    2. *User Requests* - Determine the end user requests that are to be answered by the integrated application. For each request determined, identify the tasks necessary to fulfill this request. For each of these identified tasks, identify the task agents that may be needed to be subcontractor agents for this task.
    3. *User Characteristics* - Determine the number, the geographic location, the type, and the expected workload of the end users that will be accessing the system. These characteristics are used to help determine the task agents and types of user customization that are needed for the TaMeX application.

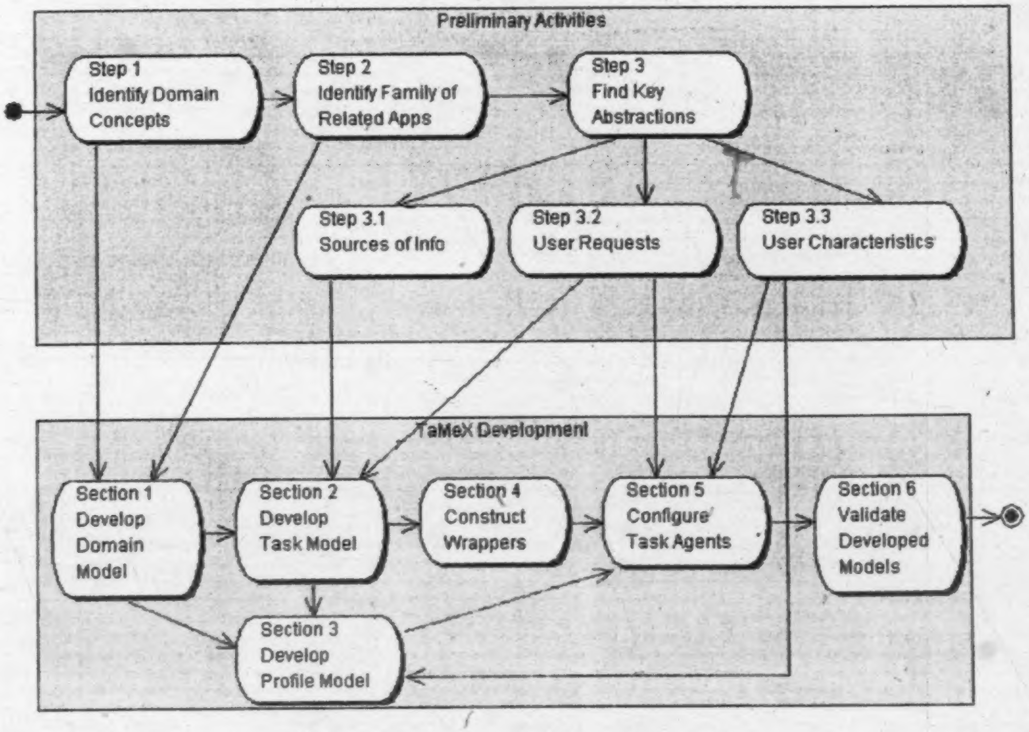

Preliminary Activities
Step 1 Identify Domain Concepts
Step 2 Identify Family of Related Apps
Step 3 Find Key Abstractions
Step 3.1 Sources of Info
Step 3.2 User Requests
Step 3.3 User Characteristics
TaMeX Development
Section 1 Develop Domain Model
Section 2 Develop Task Model
Section 4 Construct Wrappers
Section 5 Configure Task Agents
Section 6 Validate Developed Models
Section 3 Develop Profile Model

Figure 15: TaMeX Development Activities

The outputs from these preliminary steps are input to the TaMeX development process. The UML activity diagram of Figure 15 details this mapping from the preliminary activities to the main TaMeX development activities of developing the task/domain/profile models, validating these models, constructing the wrappers and configuring the application. These TaMeX development activities are described in the section of this chapter corresponding to the section number indicated in Figure 15.

The result of the TaMeX development process is an application-specific set of specifications represented in XML-based syntax. These specifications are input to the run-time environment where they drive the run-time behavior, as described in Chapter 5.

4.1 Domain Model Development

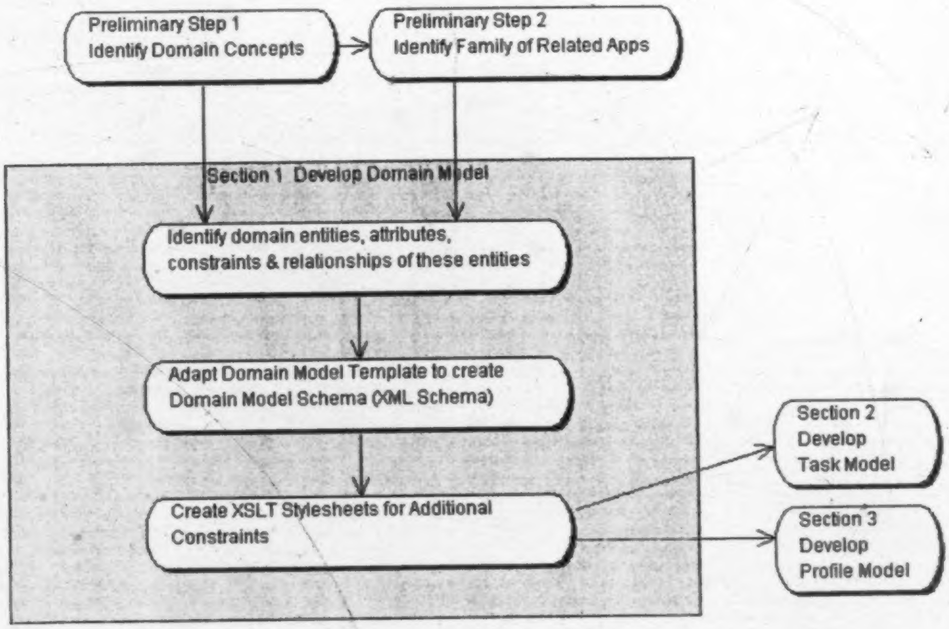

Preliminary Step 1
Identify Domain Concepts
Preliminary Step 2
Identify Family of Related Apps
Section 1 Develop Domain Model
Identify domain entities, attributes, constraints & relationships of these entities
Adapt Domain Model Template to create Domain Model Schema (XML Schema)
Create XSLT Stylesheets for Additional Constraints
Section 2 Develop Task Model
Section 3 Develop Profile Model

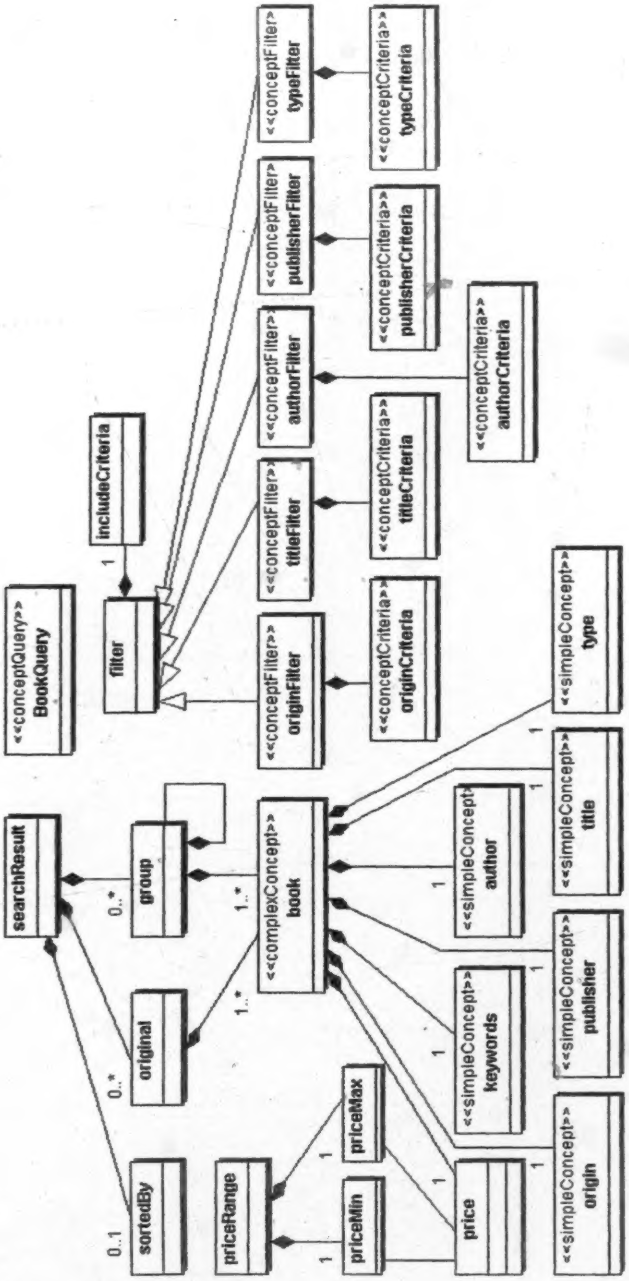
Figure 16: TaMeX Domain Model Development Activities

The first step in the application development process is for the developer to create the Domain Model that specifies the common language to be used by the integrated application - in our case, the book-finding application. As seen in Figure 16, the input to this process consists of steps 1 and 2 of the preliminary activities: identify the concepts of the particular application and identify the family of related applications to which this particular application belongs. The preliminary process identified *book*, *price*, *title*, *publisher*, and *author* as domain concepts and identified "comparative shopping" as the family of related applications. The developer expands the information for these preliminary concepts by identifying their attributes, constraints, and relationships, as well as identifying additional concepts that may be needed for this application. Some examples of additional concepts identified for the book-finding domain are the *origin* of the information source and the *type* of book (whether hardcover or softcover).

```
<!-- a single book element in the result returned by the server -->
<xsd:element name="book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="title" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="author" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="publisher" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="type" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="price" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="origin" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="keywords" minOccurs="1" maxOccurs="1"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 17: Concept Book in Book Finding Domain Model Schema

Then, using this identified information, the developer adapts the general "comparative-shopping" Domain Model Template to create the *Book-finding Domain Model Schema*. S/he accomplishes this by editing the XML schema that specifies the Domain Model Template for "comparative-shopping". The specifications for the generic concepts within this schema are left "as is" and the information for the application-specific concepts are entered in the indicated positions. For example, the template entity, *complexConcept* is replaced with the information for the concept *book*. The part of the *Book-Finding Domain Model Schema* that defines the concept *book* is shown in Figure 17. Notice that the *simpleConcept*s from the template, which compose the *complexConcept,* are replaced by the application-specific concepts of *title*, *author*, *publisher*, *type*, *origin*, and *keywords*. Thus, *book* is defined as being composed of these application-specific concepts plus the generic concept *price*. As well, each of the *simpleConcepts* that make up the *book* concept is defined as occurring exactly once.



searchResult
<<conceptQuery>> BookQuery
filter
includeCriteria
1
<<conceptFilter>> typeFilter
<<conceptFilter>> publisherFilter
<<conceptFilter>> authorFilter
<<conceptFilter>> titleFilter
<<conceptFilter>> originFilter
<<conceptCriteria>> typeCriteria
<<conceptCriteria>> publisherCriteria
<<conceptCriteria>> authorCriteria
<<conceptCriteria>> titleCriteria
<<conceptCriteria>> originCriteria
group
0..*
original
0..*
sortedBy
0..1
1..*
1..*
<<complexConcept>> book
priceRange
priceMin
priceMax
price
1
1
1
<<simpleConcept>> author
<<simpleConcept>> keywords
<<simpleConcept>> type
<<simpleConcept>> title
<<simpleConcept>> publisher
<<simpleConcept>> origin
1
1
1
1
1
1

Figure 18: Book-Finding Domain Model Schema UML Diagram

The composition of the *book* concept is shown diagrammatically in the UML representation of the Book-Finding Domain Model schema in Figure 18. The concept searchResult is another example of a schema definition for the composition of a concept. Notice that *searchResult*, the list of retrieved products from the web-based applications, is composed of *original, group*, and *sortedBy*. Both *original* and *group* can occur zero to many times. *Original* is the information as originally retrieved from the resources. *Group* is the information after the end user has arranged it by sorting, filtering, and grouping activities. *Original* and *group* are both composed of *book*s. As previously mentioned, each *book* is composed of mandatory elements such as *price*, *author*, etc.

As well as defining compositional constraints on the domain concepts, the Domain Model Schema can also define simple constraints on the value of a single field of information. XML Schema defines several facets for constraining the set of values for a data field, such as the *enumeration facet* for specifying a set of values and the *pattern facet* for specifying a pattern that the data type's values must match. In the case of the book-finding domain, only the *price* field is a candidate for value constraint checking as all the other fields are free-format. The *pattern facet* was used in the Domain Model Schema to constrain the value of the price field to be numeric with an optional '$' before the numeric price.

As previously described the Book-Finding Domain Model Schema specifies some simple constraints for the concepts, such as how they are composed. However, the more complicated constraints cannot be specified within an XML schema as shown with the price-checking constraint. Therefore, the last step in creating the domain model is for the developer to create XSL stylesheets for the complex constraints. An example in the book-finding domain of a complex constraint involving multiple concepts is the functional constraint on the *Filter by Price* task. This constraint, shown in Appendix B3, specifies a precondition on the *Filter By Price* task: namely that in order for the task to be accomplished, the values of its input parameter *priceMin* should be less than the value of *priceMax*.

The results from this process of domain development are an XML schema and a set of stylesheets that specify the entities for the book-finding domain, as well as their attributes, compositions, and constraints.

4.2 Task Model Development

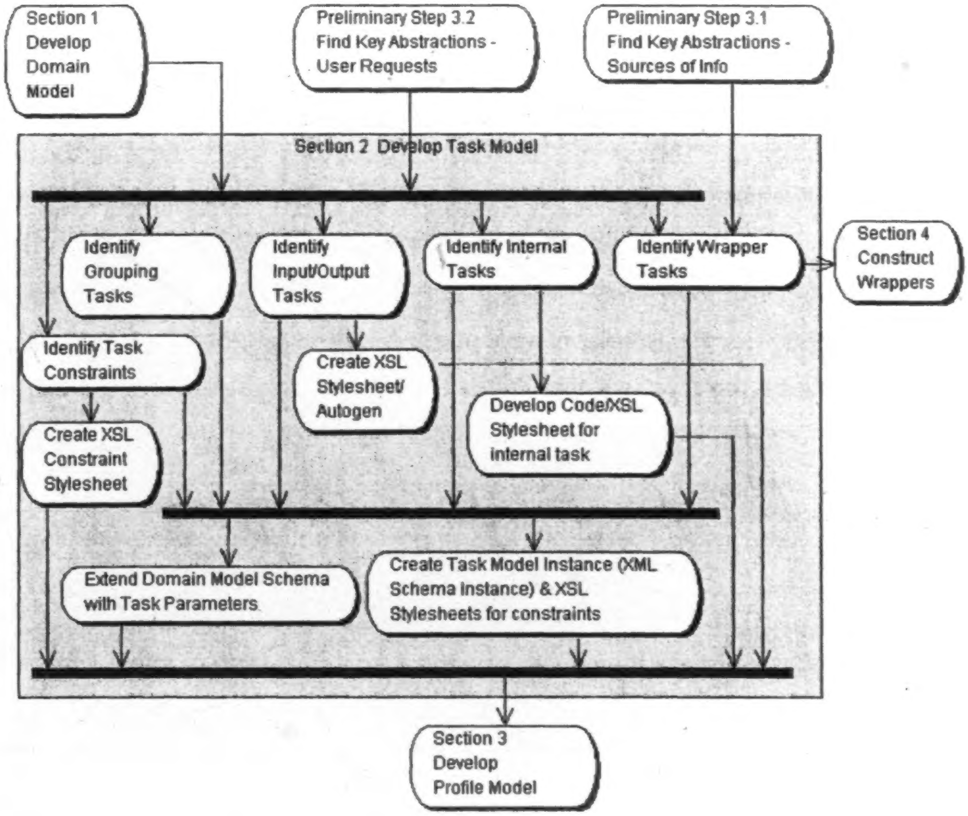

Section 1 Develop Domain Model
Preliminary Step 3.2 Find Key Abstractions - User Requests
Preliminary Step 3.1 Find Key Abstractions - Sources of Info
Section 2 Develop Task Model
Identify Grouping Tasks
Identify Input/Output Tasks
Identify Internal Tasks
Identify Wrapper Tasks
Section 4 Construct Wrappers
Identify Task Constraints
Create XSL Stylesheet/ Autogen
Develop Code/XSL Stylesheet for internal task
Create XSL Constraint Stylesheet
Extend Domain Model Schema with Task Parameters
Create Task Model Instance (XML Schema Instance) & XSL Stylesheets for constraints
Section 3 Develop Profile Model

Figure 19: TaMeX Task Model Development Activities

Once the Domain Model has been created, the developer creates the Task Model which is a hierarchical grouping of tasks and subtasks which specifies the flow of information and control in the integrated application - in our case, the book-finding application. As seen in Figure 19, the input to this process consists of steps 3.1 and 3.2 of the preliminary activities: find the key abstractions of sources of information and user requests.

The preliminary step 3.1 identified three sources of information for books: *Amazon*, *Chapters-Indigo*, and *1BookStreet*. It also identified three wrapper tasks, one for the book-finding task at each source of information. The preliminary step 3.2 identified the main user request to be answered as *'find and display book information* (such as price, title, author, publisher) *given some selection criteria'*. However, there were several variations on this main user request with regards to how the book information is to be displayed. Sometimes the information is to be sorted on one of the information fields, sometimes it is to be grouped, sometimes it is to be filtered, and sometimes a combination of sorting, grouping, and filtering is to be used for the information display. As well as the wrapper tasks previously identified, this step identified other tasks necessary to fulfill the user request such as *specifying the book selection criteria, accessing the sources of information, sorting the books, grouping the books, and filtering the books*.

Using the tasks already identified as a starting point, the developer identifies additional input, output, internal, or wrapper tasks needed, task constraints and parameters needed for these tasks, and the grouping tasks needed to structure the integrated application. For each input task identified, the developer must create an input XSL stylesheet or define an 'autogen' construct in the Task Model Instance. For each output task identified, the developer must create an XSL stylesheet to define which information will be provided as displayed output. For each internal task identified, the developer must create code and/or an XSL stylesheet to provide the functionality of the task. For each task constraint that is not structural, the developer must create an XSL stylesheet to perform the relevant functional constraint checking.

Once all the pieces needed for the Task Model are identified, the developer creates the Task Model Instance by creating an XML schema instance of the Task Model Schema which describes all the tasks identified for book-finding as well as their composition. This Task Model Instance combined with the set of book-finding XSL stylesheets of input/output presentation stylesheets, constraint-checking stylesheets, and internal task performing stylesheets comprise the model layer of the Task Model.

However, as well as creating the Task Model, the developer may also need to extend the Domain Model Schema with any task parameters that were not previously defined.

4.3 Profile Model Development

After the Domain Model and the Task Model have been developed, the developer creates the Profile Model that enables an end user to adapt the task structure according to their own preferences. As seen in Figure 20, the inputs to this process are the outputs from the two previous development steps: the Domain Model Schema and the Task Model Instance, as well as the user characteristics identified in preliminary step 3.3.

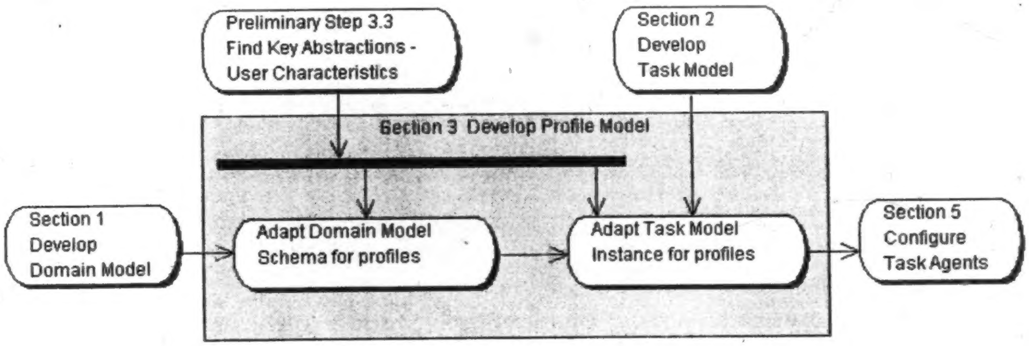

Preliminary Step 3.3
Find Key Abstractions - User Characteristics
Section 2
Develop
Task Model
Section 3 Develop Profile Model
Section 1
Develop
Domain Model
Adapt Domain Model Schema for profiles
Adapt Task Model Instance for profiles
Section 5
Configure
Task Agents

Figure 20: TaMeX Profile Model Development Activities

From the preliminary activities, the developer knows which tasks the users would like to be able to customize and in which ways. In other words, s/he knows which tasks are to be automated, removed, and/or have default input information entered - in the book-finding domain the users want to be able to enter default information to specify the search criteria and to filter the book information by price.

Using this information, the developer extends the existing Domain Model Schema with the needed concepts. S/he must also add the profile task to the Task Model Instance and add 'autogen' constructs for the input transactions that are to have default information entered - i.e. *Specify Book Selection Criteria* and *Filter by Price.*

4.4 Wrapper Construction

An example-based learning method [STS00] is used to semi-automatically construct the wrappers. Given examples of browser accesses to the web-application service of interest, the mapping between the application-domain model and the original interface of this service is learned and two generalized translation patterns are constructed.

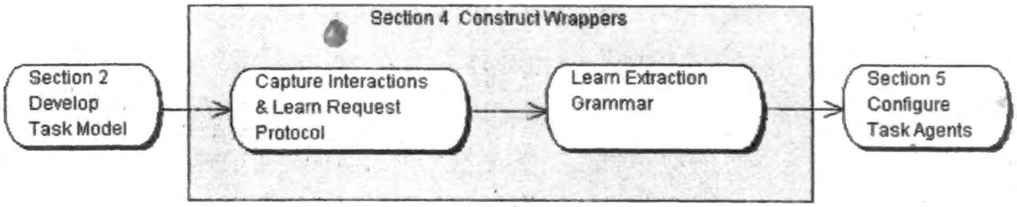

Section 4 Construct Wrappers
Section 2 Develop Task Model
Capture Interactions & Learn Request Protocol
Learn Extraction Grammar
Section 5 Configure Task Agents

Figure 21: TaMeX Wrapper Construction Activities

The rationale of this process is that, since all the web-applications to be wrapped share a common application domain, it should be possible to represent the information exchanged between the user's browser and the various web-application servers in this common application domain. In our example, the common application domain is the "book-finding domain" and the information exchanged takes the form of "HTTP-request parameters" and "HTML-response documents".

Thus, given a set of examples of accessing the same service with different parameters, the wrapper-construction process learns the mapping and constructs two generalized translation patterns. As shown in Figure 21, there are two main phases to this wrapper-construction process, with each phase creating a generalized translation pattern. The request protocol pattern, constructed in the first phase, represents how entities from the common application domain can be composed into the syntax of the HTTP request invoking the web application of interest. The extraction grammar pattern, constructed in the second phase, represents how entities from the common application domain can be extracted from the HTML response of the service. These generalized patterns exploit the tree-structure of the XML and HTML syntax and represent the mapping as correspondences between XPATH [Xpath] expressions mapping application-domain entities to web-application user-interface components.

Three wrappers were constructed for the book-finding application to correspond to the three wrapper tasks created in the Task Model. These wrappers will enable the access to the services of the three corresponding web applications. These applications are Amazon at http://www.amazon.com, Chapters-Indigo at http://www.chapters.ca, and 1BookStreet at http://www.1bookstreet.com.

4.5 Task Agent Configuration

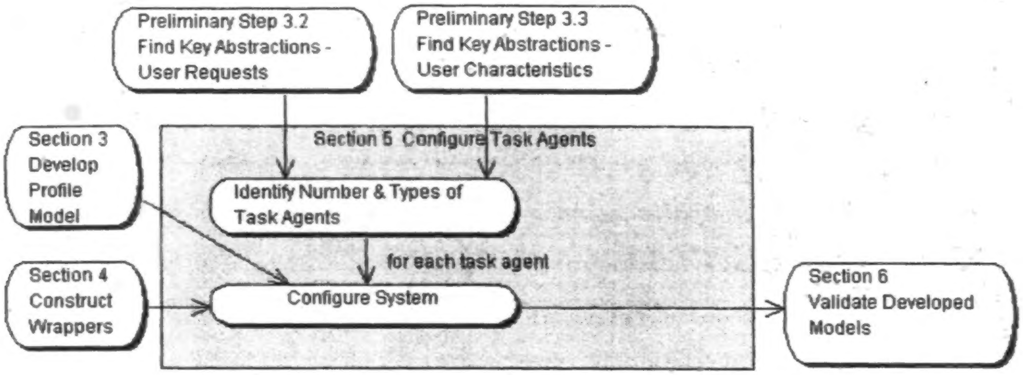

Preliminary Step 3.2
Find Key Abstractions - User Requests
Preliminary Step 3.3
Find Key Abstractions - User Characteristics
Section 3
Develop Profile Model
Section 5 Configure Task Agents
Identify Number & Types of Task Agents
for each task agent
Section 4
Construct Wrappers
Configure System
Section 6
Validate Developed Models

Figure 22: TaMeX Task Agent Configuration Development Activities

Once the models and the wrappers have been created, the developer configures the task agents necessary to run the integrated application. During the preliminary activities of finding the key abstractions, the tasks needed to fulfill the user requests were identified and the user characteristics, such as number, geographic location, and expected workload, were identified. As seen in Figure 22, this information is input to the first step of identifying the number and types of task agents. The developer uses this information, along with his/her own knowledge, to determine the number and types of task agents required. If, for example, users were expected to come from a set of distinct geographic areas, then a corresponding number of task agents would be configured to serve each of these areas. Furthermore, if an area is found to generate especially high workload, more than one task agent may be configured to serve it. Finally, due to contractual agreements, some of the wrapped web applications may allow access only from a limited number of task agents, to keep their workload low; therefore some of the task agents may be able to access these resources only indirectly.

Each of these identified task agents will share the same task structure (or a subset of it) - in our case the book-finding task structure as diagrammatically depicted in Figure 5. However, each agent has different capabilities within this task structure, including being able to access a different subset of the application's wrappers. Therefore, each task agent must be configured to its unique specifications.

There are two interfaces for accessing a task agent: a user interface and a subcontractor-agent interface. A task agent can be configured to use both interfaces or to use just one of the interfaces. If a task agent is configured to employ the user interface then the developer must include the TaMeX user interface specifications for this task agent. However s/he does not need to adapt these specifications as TaMeX employs the information specified in the declarative models for that particular application and the information produced-to-date via user interaction, along with the TaMeX user interface specifications, to generate the user interface. See Section 2.3.3 for a description of the TaMeX user interface model. To accomplish the actual configuration for each task agent, a developer must:

- *Create the Main Configuration Instance* - assign values to the various configuration options defined in the Main Configuration. This instance is an XML file that is checked for conformance to the Main Configuration Schema. See Appendix D2 for an example of the Main Configuration Instance.
- *Update the Task Model Instance* - add an *agent* element to any task to identify an alternate agent for task execution. The *agent* element contains the agent id of a "subcontractor" agent that is capable of performing the task-in-question.

The book-finding prototype application was developed using three task agents, as represented in Figure 11. There were two task agents set up to serve the two user groups of faculty (Task Agent C) and students (Task Agent A). As well as functioning as a user interface agent for students, Task Agent A was also set up to function as a subcontractor agent. Task Agent C was set up to function only as a user interface agent for faculty. The developer was concerned that "Chapters" might place a restriction on the number of agents that could access its services. In preparation for this, a third task agent (Task

Agent B) was set up that could be subcontracted by either of the other two task agents when they needed to access "Chapters". As well as being able to access "Chapters", Task Agent B was configured with the capability to access the other two web applications of "1Book" and "Amazon" so that it could function as a backup agent. Task Agent B was set up to function only as a subcontractor agent and could not be accessed by any end users. Task Agent C, the "faculty" task agent, was also configured with the capability to access "1Book" and "Amazon". However, since the students rarely needed to access "Amazon" the task agent set up to serve them, Task Agent A, was configured with the capability to access "1Book" only.

Table 10: Important Main Configuration Options for the Book Finding Application

Main Configuration Element	**Task Agent A "Student" Agent**	**Task Agent B**	**Task Agent C "Faculty" Agent**
startAgent	yes	yes	no
registeredAgents	Task Agent B	Task Agent A	Task Agent A
			Task Agent B
registeredResources	1Book	1Book	1Book
		Chapters	Amazon
		Amazon	

Table 10 shows the important Main Configuration instance options that were specified for the Book-finding application. In this table, there is a column representing each one of the task agents and a row representing each one of the important Main Configuration options. The *startAgent* option is a flag that specifies whether this task agent is to function as a subcontractor agent. It can be seen that both Task Agent A and Task Agent B are set up to function as subcontractor agents, whereas Task Agent C is not set up to function as a subcontractor agent. The *registeredAgents* option indicates an entry in the agent registry that specifies the id, name, address, and port of a task agent that is available for subcontracting by this task agent. Task Agent A and Task Agent B can only subcontract to each other, whereas Task Agent C can subcontract tasks to both Task Agent A and Task Agent B. As well as specifying the access information for each subcontracting agent in the Main Configuration instance, the actual tasks that they can perform must be specified in the Task Model instance. Task Agent A's Task Model instance specifies Task Agent B as the backup agent for the tasks that access "1Book", "Chapters", and "Amazon". Task Agent B's Task Model instance specifies Task Agent A as the backup agent for the task that accesses "1Book". Task Agent C's Task Model instance specifies Task Agent A and Task Agent B as the backup agents for the task that accesses "1Book" and Task Agent B as the backup agent for the tasks that access "Chapters" and "Amazon". The *registeredResources* option indicates an entry in the wrapper registry that specifies the detailed information about the wrappers that the task agent is configured to use. Task Agent A is configured to access "1Book", Task Agent B is configured to access all three resources, and Task Agent C is configured to access "1Book" and "Amazon".

4.6 Model Validation

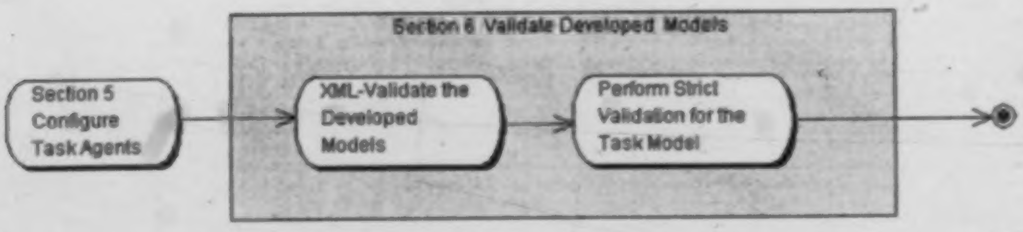

Section 6 Validate Developed Models
Section 5 Configure Task Agents
XML-Validate the Developed Models
Perform Strict Validation for the Task Model

Figure 23: TaMeX Model Validation Development Activities

After the developer has created all the models and configured the task agents, s/he must ensure that the task model and main configuration file are valid. We use "valid" to mean that the model satisfies all the constraints or rules defined by the layer above it, in this case the meta-model layer.

The first step is to check that the structural constraints of the system are being followed. This is done by checking that the XML schema instances conform to the rules in their corresponding XML schemas - i.e. the Task Model Instance and the Main

Configuration Instance conform to the Task Model Schema and the Main Configuration Schema respectively. The developer performs this validation by running TaMeX "as is" - an XML parser (Xerces) is used to check that the XML entities, their attributes and their relationships that are declared in the XML schema instances conform to the specifications of these entities in the corresponding XML schemas.

This schema validation checks that the Task Model Instance is set up properly. For example, it checks that the task model starts with a properly constructed *taskModel* element with a *name* and *start*ing task id. The *taskModel* element must contain a single *domain* element and may contain a single *modelDescription* element and one or more *task* elements. At the next level, each *task* element must have a *taskid* and must contain a *name* element and *description* element and may contain a single *conceptualConstraint* element, a single *functionalConstraint* element, multiple *agent* elements, multiple *link* elements, and multiple *clearActivation* elements. As well as the rules just listed that are applicable to all *task* elements there are also constraints that are applicable to specific task-types. For example, an *inputTask* element must contain a single *output* element and may contain multiple *var* elements, an *input* element, and either an *autogen* element or a *xslt* element. A *wrapper* element, however, must contain a single *input* element, a single *output* element, and a single *wrapper* element and may contain multiple *var* elements.

Unfortunately, not all the rules for validating the models can be expressed in an XML schema. For example, in the task model we want to ensure that the subtask elements of a grouping task reference valid tasks within that same task model. We also want to ensure that for each wrapper used in the task model there is a corresponding registered resource with the same name inside the main configuration file. For these types of rules, we implemented *strict validation*, which validates the task model with the extra rules that couldn't be specified using an XML schema. The validation rules that we added are shown in Table 11.

The strict validation rules are specified in an XSL stylesheet and the location of this stylesheet is specified in the *strictValidationXSL* element of the Main Configuration Instance. Strict validation applies this stylesheet to the Task Model Instance xml file to perform the actual validation. The Strict Validation Stylesheet used in TaMeX is shown in Appendix F1.

Table 11: Strict Validation Rules

Include the Main Configuration File?	**Strict Validation Rule Description**
Not needed	• check that all of the task ids within a task model are unique
	• check that task ids do not contain quote marks " ' " or pipe " | " characters
	• check that the external Ids on the task statements that can be executed by an external agent are properly formatted.
	• check that the start id of the task model points at a valid task id
	• check that the subtask elements of a grouping task reference valid tasks within that same task model
	• check that the link and clearActivation elements reference valid tasks that exist within the task model
	• check that variable names and infoIn names (from which variables will be constructed internally as the data is loaded) do not contain double quote marks
	• check that variable names are unique to each "var" declaration inside a task
	• check that infoIn names are unique to each infoIn declaration inside a task
	• check that there are no defined variable's names that appear in an infoIn statement. This is to ensure that infoIn statements cannot override variables declared with the "var" declaration
	• check that each variable used in a task has been defined (using either a "var declaration or an "infoIn" declaration)
Main Configuration File **MUST** be included for these checks	• check that for each wrapper used in the task model there is a corresponding registered resource with the same name inside the Main Configuration File
	• check that for each agent used in the task model there is a corresponding agent (same agent id) that has been configured in the Main Configuration File

Once the developer has successfully XML-validated the models, s/he must request the TaMeX system to perform strict validation on the task model. This request is necessary since XML validation only is the default because strict validation is very time-consuming to run. The developer requests strict validation by specifying the appropriate *validation* attribute of the appropriate task model within the *registeredTaskModels* in the Main Configuration Instance to be "strict" instead of the default of "normal".

Chapter 5 Run-time Behavior

To illustrate the run-time behavior of the aggregate applications developed in the TaMeX multi-agent framework, we will use as an example our book-finding prototype described in Section 2.1. This prototype was built to support explorative comparative book shopping and integrates the services provided by three different book-selling web-based applications.

In the scenario described in this chapter, the instructor wants to decide on a textbook to use in her new course on "Java programming". She will use the TaMeX book-finding application to collect information relevant to making a decision. She will contact Task Agent C as she has been told by a colleague that Task Agent C can run the book-finding application. See Figure 11 for the component architecture for the book-finding application, showing all the task agents that are involved. The other task agents involved are not known to the instructor.

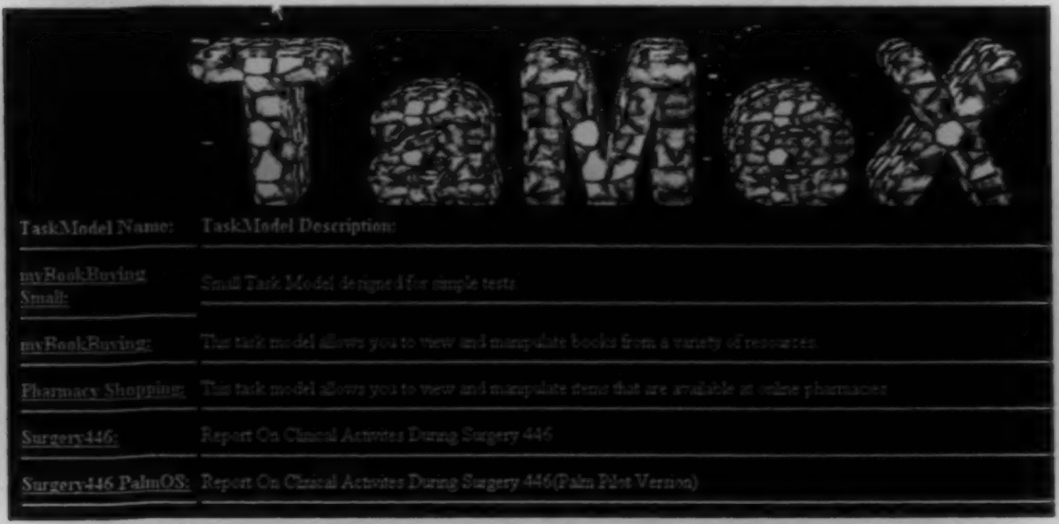
This chapter illustrates the run-time behavior of TaMeX step-by-step, starting with the instructor's selection of the task model driven application desired. Following the selection of the task model, the instructor's process of manually selecting one task at a time and executing it is described. The steps that she takes in this process are starting up the selected application, specifying the book selection criteria, accessing *Amazon* - one of the web-based resources, sorting the results from *Amazon*, and viewing the results that are returned automatically after the sort. When she views the results, the instructor realizes that she meant to access all three of the book-selling web-based applications and then sort the results by price.

To streamline the search procedure and save her the tedium of selecting and executing one task at a time, she decides to customize the task model and automate many of the tasks. The last sections of this chapter describe her customization of the task model and both her own and the task agent's process of using this customized task model. During the task agent's execution of the customized task model, it found that it needed to collaborate with another task agent. This collaboration is described in Sections 5.9 and

5.11.

5.1 Task Model Selection

After the initial contact by the instructor, Task Agent C returns the *task model selection* screen as shown in Figure 24. The instructor reads the task model descriptions and locates the *myBookBuying* task model via its description, "This task model allows you to view and manipulate books from a variety of resources". She then clicks on *myBookBuying* to select this task model.

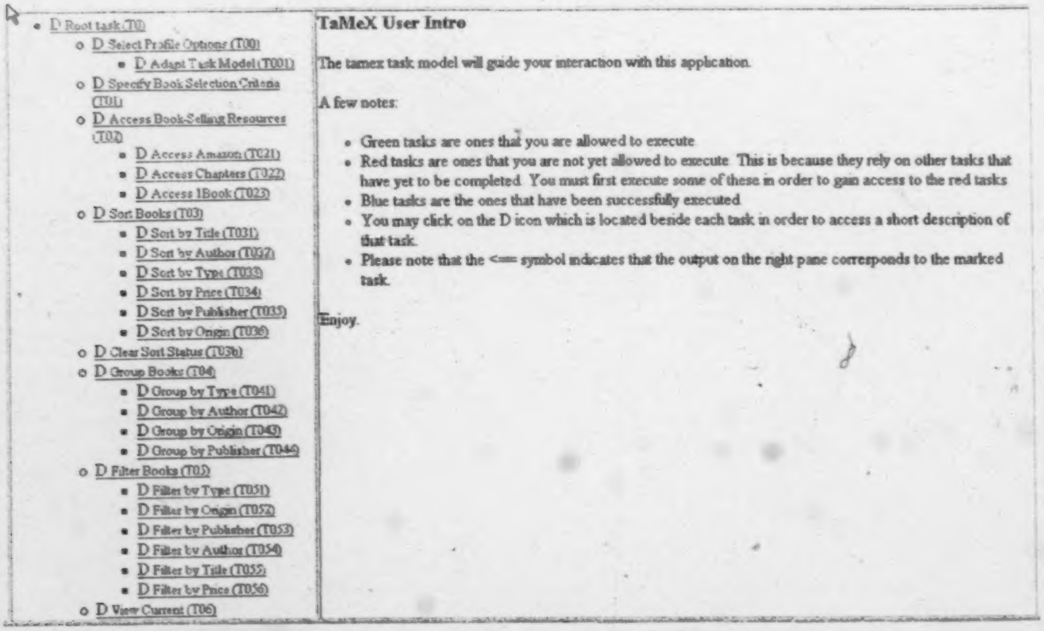


TaskModel Name:	TaskModel Description:
myBookBuying Small:	Small Task Model designed for simple tests
myBookBuying:	This task model allows you to view and manipulate books from a variety of resources
Pharmacy Shopping:	This task model allows you to view and manipulate items that are available at online pharmacies
Surgery446:	Report On Clinical Activites During Surgery 446
Surgery446 PalmOS:	Report On Clinical Activites During Surgery 446(Palm Pilot Version)

Figure 24: Task Model Selection Screen

5.2 Selected Task Model Start-up

After receiving the *myBookBuying* selection, Task Agent C returns the first *task model interaction screen* to the instructor. This screen, as shown in Figure 25, consists of two sections: the *hierarchical task menu* for the *myBookBuying* task model on the left and the *TaMeX user intro* description in the data area on the right. Notice that the hierarchical task menu reflects the task structure for book-finding that is depicted in Figure 5. The top-most task, "*find book information*" is the root of the menu. Its subtasks, "*specify book selection criteria*", "*access book-selling resources*" etc. are the second level menu options, and finally the executable leaf tasks are the options at the third level of the menu tree.



- D Root task (T0)
    - D Select Profile Options (T00)
        - D Adapt Task Model (T001)
    - D Specify Book Selection Criteria (T01)
    - D Access Book-Selling Resources (T02)
        - D Access Amazon (T021)
        - D Access Chapters (T022)
        - D Access 1Book (T023)
    - D Sort Books (T03)
        - D Sort by Title (T031)
        - D Sort by Author (T032)
        - D Sort by Type (T033)
        - D Sort by Price (T034)
        - D Sort by Publisher (T035)
        - D Sort by Origin (T036)
    - D Clear Sort Status (T03b)
    - D Group Books (T04)
        - D Group by Type (T041)
        - D Group by Author (T042)
        - D Group by Origin (T043)
        - D Group by Publisher (T044)
    - D Filter Books (T05)
        - D Filter by Type (T051)
        - D Filter by Origin (T052)
        - D Filter by Publisher (T053)
        - D Filter by Author (T054)
        - D Filter by Title (T055)
        - D Filter by Price (T056)
    - D View Current (T06)

TaMeX User Intro

The tamex task model will guide your interaction with this application.

A few notes:

- Green tasks are ones that you are allowed to execute.
- Red tasks are ones that you are not yet allowed to execute. This is because they rely on other tasks that have yet to be completed. You must first execute some of these in order to gain access to the red tasks.
- Blue tasks are the ones that have been successfully executed.
- You may click on the D icon which is located beside each task in order to access a short description of that task.
- Please note that the <== symbol indicates that the output on the right pane corresponds to the marked task.

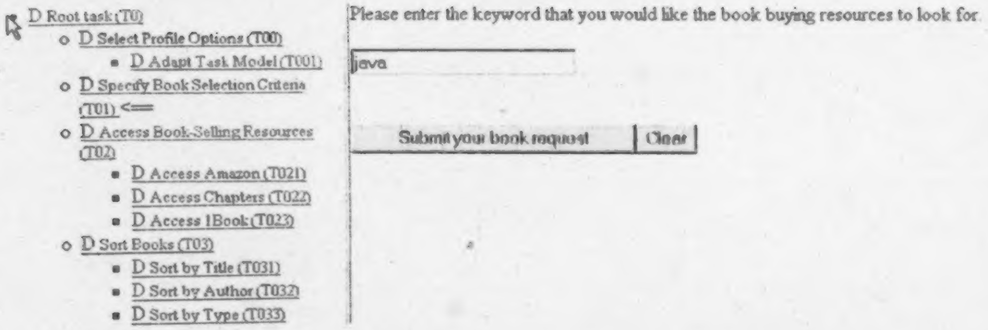
Enjoy.

Figure 25: First Task Model Interaction Screen

Notice also that the only "green" task (i.e. executable task) in this menu is the root task. The instructor starts the execution of the book-finding task model by clicking on this root task, which corresponds to the top-most task "*find book information*" in the book-finding application.

5.3 Input Task Execution

As can be seen in Figure 26, Task Agent C returns a hierarchical menu with two more tasks that are now executable: *Select Profile Options (T00)* and *Specify Book Selection Criteria (T01)*. At this point, the instructor does not want to select any profile options - she is happy with interacting with the current task model. So she clicks on the user-interaction (input) task of *Specify Book Selection Criteria (T01)*.



- D Root task (T0)
  - D Select Profile Options (T00)
    - D Adapt Task Model (T001)
  - D Specify Book Selection Criteria (T01) <==
  - D Access Book-Selling Resources (T02)
    - D Access Amazon (T021)
    - D Access Chapters (T022)
    - D Access 1Book (T023)
  - D Sort Books (T03)
    - D Sort by Title (T031)
    - D Sort by Author (T032)
    - D Sort by Type (T033)

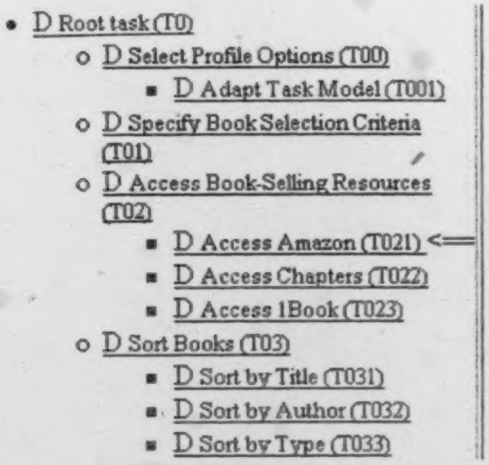
Please enter the keyword that you would like the book buying resources to look for.

java

Figure 26: Specify Book Selection Criteria Screen

This task becomes the first leaf task executed by Task Agent C. Its purpose is to enable the instructor to specify the problem at hand. Task Agent C sends the *query* entry form, which is generated by the XSL stylesheet corresponding to this input task, to the instructor's browser. Using this form, the instructor types in the keyword of "java" in the entry field for the books she wants to find.

5.4 Wrapper Task Execution



- D Root task (T0)
  - D Select Profile Options (T00)
    - D Adapt Task Model (T001)
  - D Specify Book Selection Criteria (T01)
  - D Access Book-Selling Resources (T02)
    - D Access Amazon (T021) <==
    - D Access Chapters (T022)
    - D Access 1Book (T023)
  - D Sort Books (T03)
    - D Sort by Title (T031)
    - D Sort by Author (T032)
    - D Sort by Type (T033)

Figure 27: Access Amazon Screen (after task completed)

After the desired book selection criteria have been specified, Task Agent C proceeds to accomplish the next task, i.e., to access the book-selling resources (T02) via the wrapped web-based applications. This grouping task is decomposed into a set of information-collection (wrapper) tasks, one for each wrapped web-based application. In this example there are three wrapper tasks, *Access Amazon (T021)*, *Access Chapters (T022)*, and *Access 1Book (T023)* corresponding to accessing the three wrapped applications, Amazon, Chapters-Indigo and 1BookStreet. Each wrapper task is accomplished by invoking a request to the task-specified wrapped web-based application. This process is started when a user selects a wrapper task from the menu. The task agent responds by sending a request to the particular wrapper of the selected application that will produce the desired output given the already-entered set of inputs.

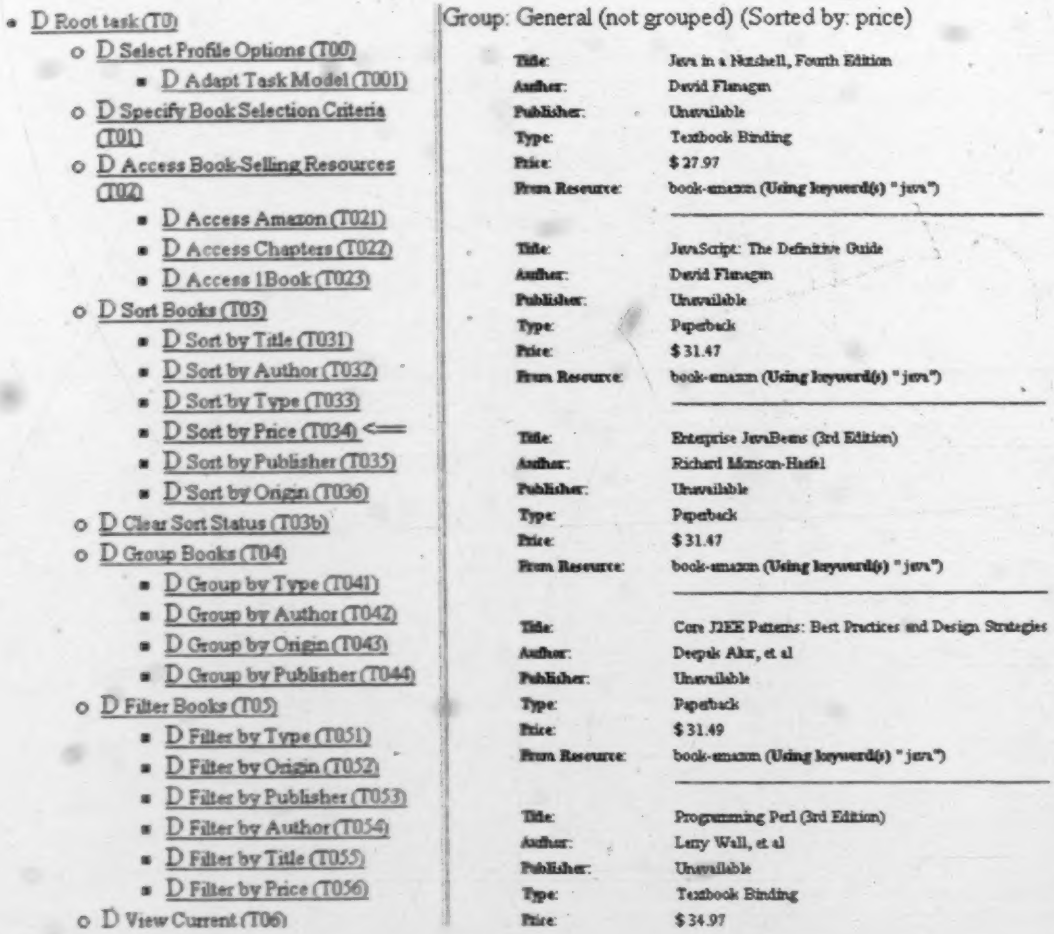
In our example, instructor selects the first wrapper task of *Access Amazon (T021)*. When the instructor selects the first wrapper task of *Access Amazon (T021)* Task Agent C responds directly (since Task Agent C is capable of performing this task) by sending a request to the particular wrapper of the Amazon application that produces a list of books given a keyword. The keyword that is used for this access was previously entered by the instructor in the *Specify Book Selection Criteria (T01)* task and is "java". Therefore, in this case, the T021 task wrapper will return a list of all the books with a keyword of "java" that can be found at the Amazon website. However, as shown in Figure 27, Task Agent C only returns an indication that *Access Amazon (T021)* has been successfully executed. It is up to the instructor to select a task to view the list of books or to select tasks to arrange the books before viewing.

5.5 Internal Task Execution

Once a resource has been accessed, in this case Amazon, the top-level task in the hierarchical menu for the arranging tasks (sort, group, filter) are now "green" to show that they are executable. The leaf tasks of these arranging tasks (*Sort Books (T03)*, *Group Books (T04)*, *Filter Books (T05)*) are internal tasks. The instructor can perform them iteratively and in any order until she has the selection of books to her liking. She can choose to display the currently selected and arranged books at any time by performing *View Current (T06)*. The *Sort Books (T03)* grouping task allows the user to choose any of the following six sorting options tasks: *by title*, *by author*, *by type*, *by price*, *by publisher*, or *by origin*. The *Group Books (T04)* and *Filter Books (T05)* grouping tasks offer a similar set of optional subtasks.

In our example, the instructor wants to sort the books by their price. Thus, she selects the *Sort by Price (T034)* internal task from her browser's menu after selecting the *Sort Books (T03)* grouping task. Task Agent C performs the sort internal task and stores the resulting list of "java" books sorted by price.

5.6 Output Task Execution



- D Root task (T0)
    - D Select Profile Options (T00)
        - D Adapt Task Model (T001)
    - D Specify Book Selection Criteria (T01)
    - D Access Book-Selling Resources (T02)
        - D Access Amazon (T021)
        - D Access Chapters (T022)
        - D Access 1Book (T023)
    - D Sort Books (T03)
        - D Sort by Title (T031)
        - D Sort by Author (T032)
        - D Sort by Type (T033)
        - D Sort by Price (T034) <==
        - D Sort by Publisher (T035)
        - D Sort by Origin (T036)
    - D Clear Sort Status (T03b)
    - D Group Books (T04)
        - D Group by Type (T041)
        - D Group by Author (T042)
        - D Group by Origin (T043)
        - D Group by Publisher (T044)
    - D Filter Books (T05)
        - D Filter by Type (T051)
        - D Filter by Origin (T052)
        - D Filter by Publisher (T053)
        - D Filter by Author (T054)
        - D Filter by Title (T055)
        - D Filter by Price (T056)
    - D View Current (T06)

Group: General (not grouped) (Sorted by: price)

Title: Java in a Nutshell, Fourth Edition
Author: David Flanagan
Publisher: Unavailable
Type: Textbook Binding
Price: $ 27.97
From Resource: book-amazon (Using keyword(s) "java")

Title: JavaScript: The Definitive Guide
Author: David Flanagan
Publisher: Unavailable
Type: Paperback
Price: $ 31.47
From Resource: book-amazon (Using keyword(s) "java")

Title: Enterprise JavaBeans (3rd Edition)
Author: Richard Monson-Haefel
Publisher: Unavailable
Type: Paperback
Price: $ 31.47
From Resource: book-amazon (Using keyword(s) "java")

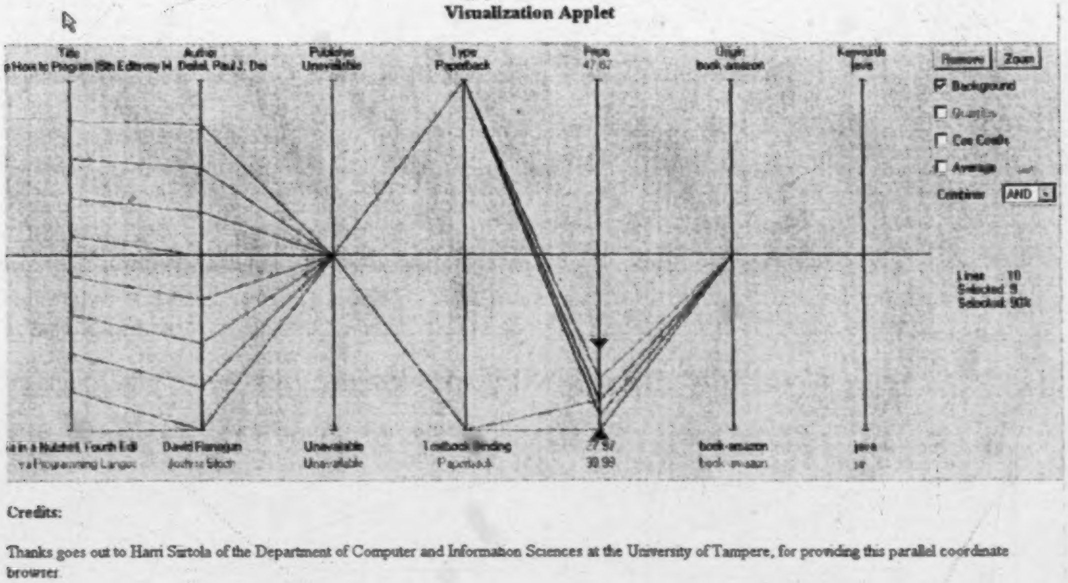
Title: Core J2EE Patterns: Best Practices and Design Strategies
Author: Deepak Alur, et al
Publisher: Unavailable
Type: Paperback
Price: $ 31.49
From Resource: book-amazon (Using keyword(s) "java")

Title: Programming Perl (3rd Edition)
Author: Larry Wall, et al
Publisher: Unavailable
Type: Textbook Binding
Price: $ 34.97

Figure 28: Sort by Price Screen (after task completed)

As the list step of the *sort* task, Task Agent C automatically executes the output task of *View Current (T06)* to display these sorted books to the instructor. The result of this output user-interaction task is shown to the right of Figure 28. Task Agent C generated this output display by applying the XSL stylesheet associated with the *View Current (T06)* task to the "results" domain concept (i.e. the list of "java" books sorted by price) stored in its DOM.

The instructor would like a visual representation of the results. She would particularly like to see how many books are less than $50.00. She selects the *Show Explorer Applet (T09)* task. This is an output task that makes use of the parallel coordinate browser applet developed by Harri Siirtola [Siir03]. The first display the instructor sees has no selected lines on it because she has not yet entered a category. She selects the lower portion of the *Price* category. The XSL stylesheet associated with the *Show Explorer Applet (T09)* task is used to generate the applet display, as shown in Figure 29. The instructor can see visually that 90% of the books (9 out of 10) cost less than $47.07.


Visualization Applet
Title
Author
Publisher
Unavailable
Type
Paperback
Price
47.67
Origin
book-amazon
Keywords
java
Remove
Zoom
☑ Background
☐ Quartiles
☐ Cor Coeffs
☐ Average
Combine
AND
Lines 10
Selected 9
Selected 90%
David Flanagan
Unavailable
Textbook Binding
27.97
30.99
book-amazon
java

Credits:

Thanks goes out to Harri Siirtola of the Department of Computer and Information Sciences at the University of Tampere, for providing this parallel coordinate browser.

Figure 29: Show Explorer Applet Screen

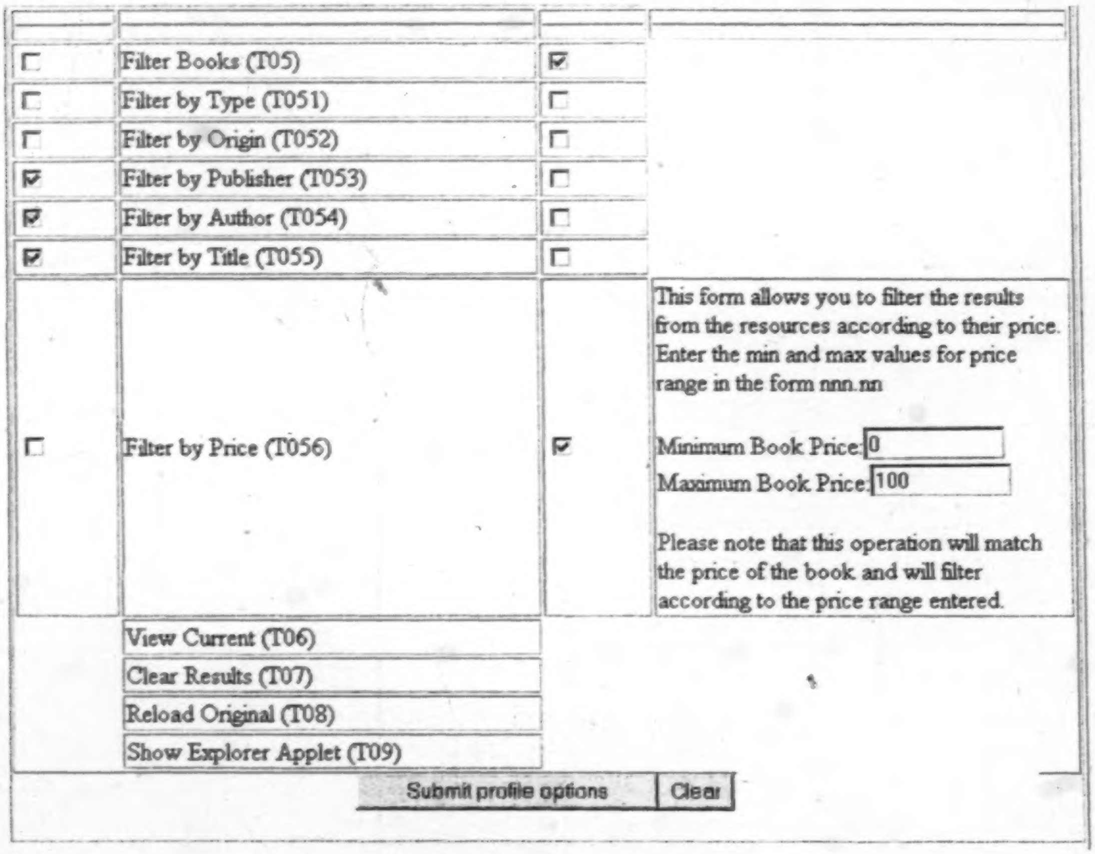
5.7 User Profile Customization of the Task Model

The instructor realizes that she only accessed one book resource and she wants to check out the books from all the available resources. She also wants to sort the books by price and filter out all the books that cost more than $100.00. Since she knows the tasks she would like to execute and since she does not want to repeat the tedious process of selecting one task at a time, the instructor decides to modify the task model to suit her requirements and automate the tasks necessary for her particular query. As well as automating the book-finding process, the instructor would like to simplify the hierarchical task menu displayed on the task interaction screen. She does not like having to scroll down to see all the tasks in the menu so she decides to remove all the unneeded tasks from the task model.

First, the instructor restarts the task model by clicking on *Restart this task model* under the general navigation options at the bottom of the hierarchical task menu. Task Agent C returns the starting *task model interaction* screen with only the *Root task (T0)* executable. The instructor selects the *Root task (T0)* and Task Agent C returns a *task model interaction* screen with the additional tasks of *Select Profile Options (T00)* and *Specify Book Selection Criteria (T01)* executable.

This time the instructor clicks on the *Select Profile Options (T00)* task instead of the *Specify Book Selection Criteria (T01)* task since she wants to customize the application to streamline her request for "java" books. She then selects the *Adapt Task Model (T001)* task from the returned *task model interaction* screen.

Task Agent C automatically generates the Adapt Task Model screen, shown in Figure 14, by applying a stylesheet to the task model. The instructor starts filling in this input form by checking the boxes in the *Automate* column beside the tasks that she wants to automate: *Specify Book Selection Criteria (T01), Access Amazon (T021), Access Chapters (T022), Access 1Book (T023), Sort Books (T03), Sort by Price (T034), Filter Books (T05),* and *Filter by Price (T056).* She then enters the default information for the two input tasks that she automated: she enters the keyword "java" beside the *Specify Book Selection Criteria (T01)* task and she enters a *Minimum Book Price* of 0 and a *Maximum Book Price* of 100 beside the *Filter by Price (T056)* task. Finally she decides to remove all the tasks that she does not use so that she can see the entire task model menu on one screen. She does this by checking the boxes in the Remove column beside the tasks that she wants to remove: *Select Profile Options (T00)*, *Sort by Title (T031)*, *Sort by Author (T032)*, *Sort by Type (T033)*, *Sort by Publisher (T035)*, *Sort by Origin (T036)*, *Group by Author (T042)*, *Group by Publisher (T044)*, *Filter by Publisher (T053)*, *Filter by Author (T054)*, and *Filter by Title (T055)*.



☐	Filter Books (T05)	☑	
☐	Filter by Type (T051)	☐	
☐	Filter by Origin (T052)	☐	
☑	Filter by Publisher (T053)	☐	
☑	Filter by Author (T054)	☐	
☑	Filter by Title (T055)	☐	
☐	Filter by Price (T056)	☑	This form allows you to filter the results from the resources according to their price. Enter the min and max values for price range in the form nnn.nn Minimum Book Price: 0 Maximum Book Price: 100 Please note that this operation will match the price of the book and will filter according to the price range entered.
	View Current (T06)		
	Clear Results (T07)		
	Reload Original (T08)		
	Show Explorer Applet (T09)		

Figure 30: Instructor Filled-in Adapt Task Model Screen (last portion)

Figure 30 shows the last portion of the filled-in Adapt Task Model screen and shows the profile options filled in by the instructor for the filter tasks. Notice that all three types of user profile customizations are depicted in this figure. A summary of the instructor's profile customization of the book-finding application is shown in Table 12.

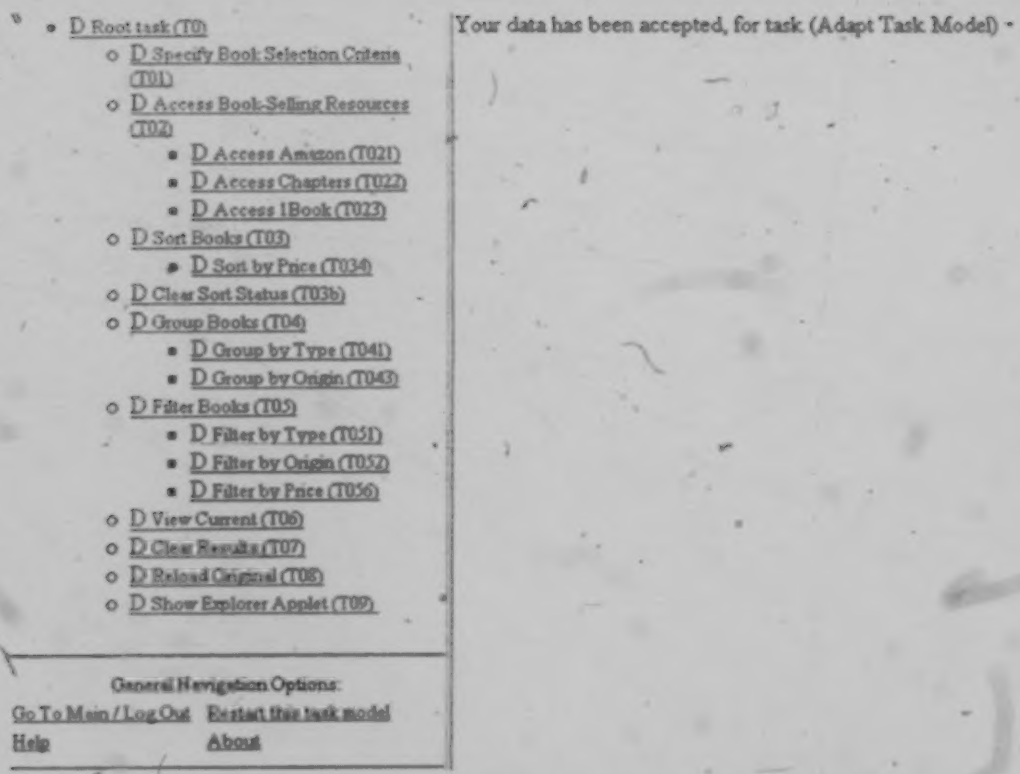
The instructor has now filled in all the profile options that she wants, so she clicks on the *Submit profile options* button.

Table 12: Instructor Profile Customization of the Book-finding Application

Category	Instructor Customization
Task Structure Customization	-removed all the profile options tasks -removed the grouping tasks 'Group by Author' and 'Group by Publisher' -removed all the sorts except 'Sort by Price' -removed the filtering tasks 'Filter by Publisher', 'Filter by Author' and 'Filter by Title'
Execution Control	-chose to automatically specify the book selection criteria -chose to automatically access the book information from all three wrapped web-based applications -chose to automatically sort the retrieved books by price -chose to automatically filter the retrieved books by price
Default Value Definition	-defined default value of "java" for the criteria to be used for book selection -defined default values for a $0 - $100 price range of books to be filtered by price

5.8 Task Agent-Driven Execution of the Customized Task Model

Task Agent C uses the execution control and task structure customization profile options that the instructor has just entered to adapt the task model to reflect the instructor's desired changes. For each *remove* and *automate* box checked by the instructor a Profile Adaptation stylesheet is applied to the current task model to modify it to implement the selected change. The removeTask.xsl stylesheet, shown in Appendix C3, and the automateTask.xsl stylesheet, shown in Appendix C2 are used for the removal and automate processes respectively. Note that the applications are successive so that each stylesheet application works on the task model that was created by the previous stylesheet application. The default value definition options, which specify the default values for input to tasks that will be requested later, do not change the task model. They are saved in Task Agent C's session DOM for later use by the appropriate input task. Figure 31 shows the display returned to the instructor after Task Agent C has successfully completed the adaptation of the task model. Notice that the hierarchical task menu on the left side of the figure is much simpler than the previous multi-screen display.


D Root task (T0)
D Specify Book Selection Criteria (T01)
D Access Book-Selling Resources (T02)
D Access Amazon (T021)
D Access Chapters (T022)
D Access 1Book (T023)
D Sort Books (T03)
D Sort by Price (T034)
D Clear Sort Status (T03b)
D Group Books (T04)
D Group by Type (T041)
D Group by Origin (T043)
D Filter Books (T05)
D Filter by Type (T051)
D Filter by Origin (T052)
D Filter by Price (T056)
D View Current (T06)
D Clear Results (T07)
D Reload Original (T08)
D Show Explorer Applet (T09)
General Navigation Options:
Go To Main / Log Out
Restart this task model
Help
About

Your data has been accepted, for task (Adapt Task Model) -

Figure 31: Instructor-Customized Hierarchical Task Menu

The instructor clicks on *Root task (T0)*. However, since she has automated all the required tasks, Task Agent C will do all the work and return a list of "java" books that are sorted and filtered by price ($0-$100 price range) to her browser. She does not need to enter anything else before receiving her list of requested books.

Task Agent C now continues its processing using the newly-configured task model as its driver. It traverses this instructor-modified task model searching for executable tasks and executes immediately any 'automatic' executable tasks that it finds.

The first 'automatic' executable task that it encounters is *Specify Book Selection Criteria (T01)*. However, since the instructor has already entered default information for this input task, Task Agent C obtains the user input from the default location of its session DOM instead of sending the *query* entry form to the instructor's browser.

After obtaining the book selection criteria, Task Agent C proceeds to accomplish the next task, i.e., to access the book-selling resources (T02) via the wrapped web-based applications. Since all three of the wrapper tasks have been 'automated' by the instructor, Task Agent C invokes a request to each of these task-specified wrapped web-based applications without needing any further user input. Task Agent C responds directly to the first wrapper task of *Access Amazon (T021)* since it is capable of performing this task. It sends a request for "java" books to the particular wrapper of the Amazon application that produces a list of books given a keyword. The T021 task will return a list of all the books with a keyword of "java" that can be found at the Amazon website.

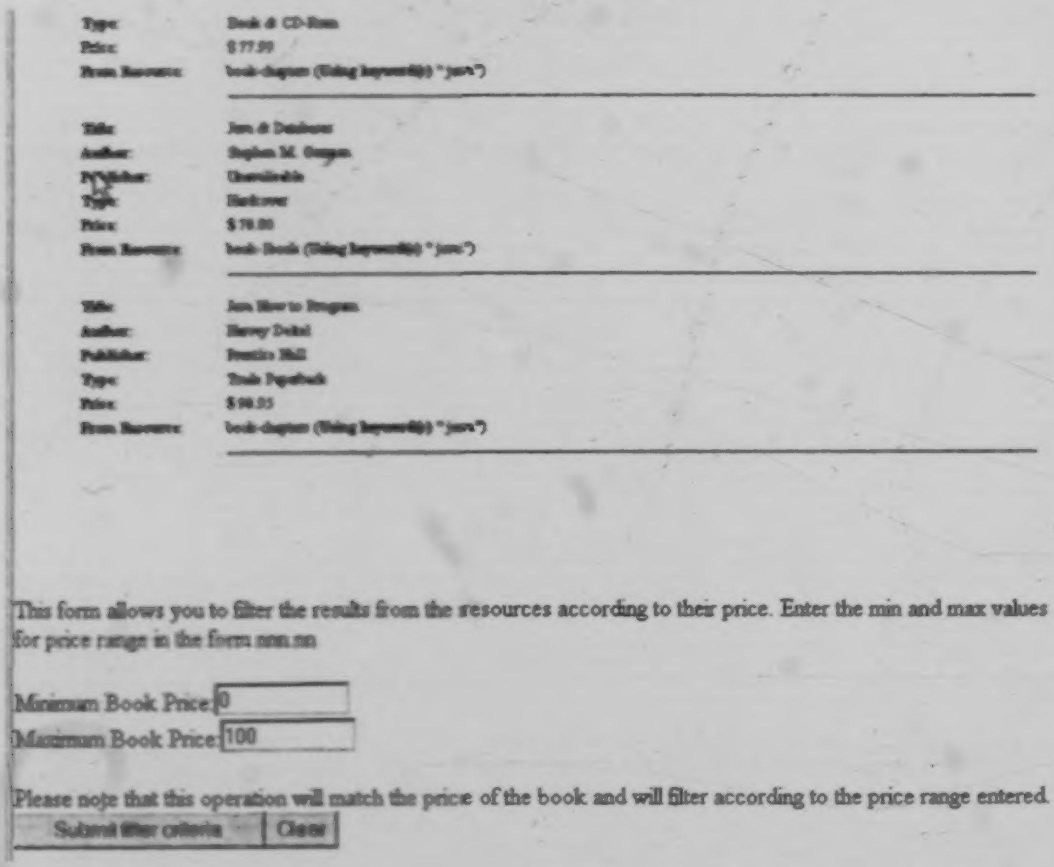
5.9 Collaboration with Another Task Agent

Task Agent C attempts to automatically execute the next wrapper task of *Access Chapters (T022)*. However, this time Task Agent C fails at this task because it does not have the capability to access the Chapters-Indigo website. One reason that Task Agent C may not have the capability is because it is geographically far away from the Chapters-Indigo application, thus making the communication between them inefficient. Task Agent C takes corrective action and identifies that Task Agent B is capable of accessing the Chapters-Indigo service. After having identified Task Agent B, Task Agent C begins its collaboration with this agent by sending Task Agent B a request to perform this particular task activity along with the variables needed to perform it. Task Agent B executes this activity by sending a request for "java" books to the particular wrapper of the Chapters-Indigo application that produces a list of books given a keyword. After receiving the results, Task Agent B returns this list of books to the originating Task Agent C. Agent C has now completed the *Access Chapters (T022)* task. Task Agent C responds directly to the third wrapper task of *Access 1Book (T023)* since it is capable of performing this task. It sends a request for "java" books to the particular wrapper of the 1Book application that produces a list of books given a keyword and receives the results from 1Book. Task Agent C now has a list of "java" books from all three of the web-based applications.

The next 'automatic' executable tasks that Task Agent C encounters are *Sort Books (T03)* with its 'automatic' subtask *Sort by Price (T034)* and *Filter Books (T05)* with its 'automatic' subtask *Filter by Price (T056)*. Task Agent C sorts the list of "java" books from all three sites by their price and then filters this list by the price range of $0 to

$100. Task Agent C obtained this price range to use for the filtering from the default information that the instructor previously entered on the *Adapt Task Model (T001)* profile input form and that was now saved in the default location of its session DOM.

Since there are no more 'automatic' executable tasks for Task Agent C to execute it returns its results to the instructor's browser. Figure 32 shows the last screen display of these results: the end of the sorted and filtered list of "java" books followed by the *Filter by Price* default price range values that were used to do the filtering.



This form allows you to filter the results from the resources according to their price. Enter the min and max values for price range in the form nnn.nn

Minimum Book Price: 0
Maximum Book Price: 100

Please note that this operation will match the price of the book and will filter according to the price range entered.

Figure 32: Task Agent-Driven Execution of the Customized Task Model Results Screen

5.10 User-Driven Execution of the Customized Task Model

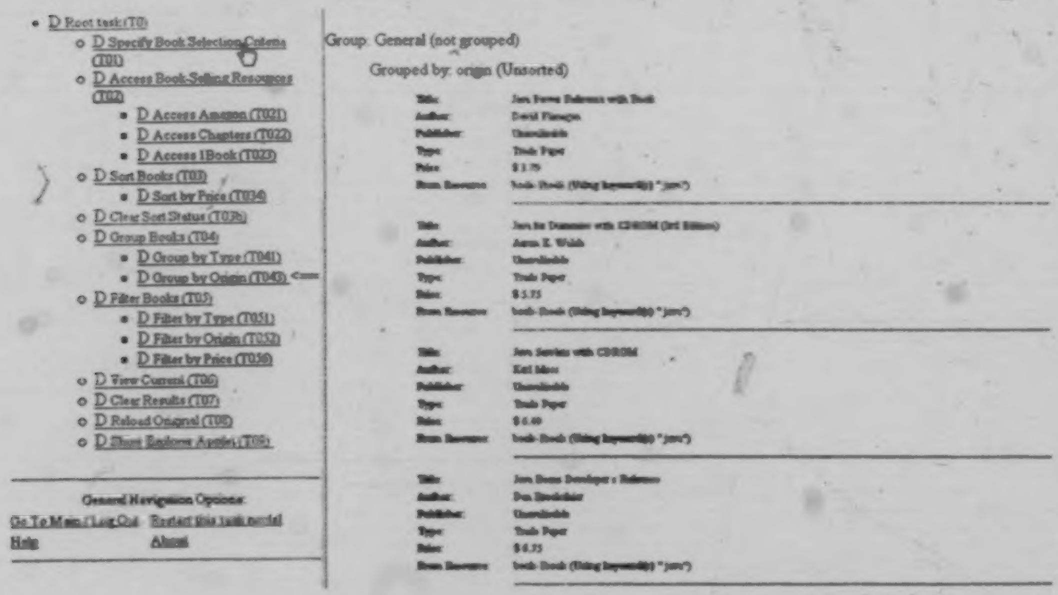

D Root task (T0)
D Specify Book Selection Criteria (T01)
D Access Book-Selling Resources (T02)
D Access Amazon (T021)
D Access Chapters (T022)
D Access IBook (T023)
D Sort Books (T03)
D Sort by Price (T034)
D Clear Sort Status (T03b)
D Group Books (T04)
D Group by Type (T041)
D Group by Origin (T043)
D Filter Books (T05)
D Filter by Type (T051)
D Filter by Origin (T052)
D Filter by Price (T056)
D View Current (T06)
D Clear Results (T07)
D Reload Original (T08)
D Show Explorer Applet (T09)
General Navigation Options:
Go To Main / Log Out
Restart this task model
Help
About
Group: General (not grouped)
Grouped by: origin (Unsorted)
Title:
Author:
Publisher:
Type:
Price:
From Resource:

Figure 33: Group by Origin Screen (after task completed)

The instructor has received the answer to her request, a list of "java" books from all three web-based applications sorted by price and filtered by a price range of $0 to $100. Note that she accomplished this by one click to her customized task model hierarchy. However, she can continue interacting with TaMeX if the results are not to her satisfaction.

Since the instructor gets a 10% discount at Amazon and special promotional rates at Chapters-Indigo, she would like to see this list grouped by origin so that she can more easily determine the least expensive place to buy her textbook. She selects the *Group Books (T04)* task followed by the *Group by Origin (T043)* task from her simplified and customized task menu hierarchy. Task Agent C returns the results as shown in Figure 33.

5.11 Collaboration with Another Task Agent upon Failure

Also the instructor wants to group the books by their type, i.e., hardcover or paperback, to help determine which book is a better value. Thus, she selects the "Group by Type (T041)" internal task from her browser's menu. Task Agent C fails at performing this task directly because its XSL stylesheet needed to perform the grouping function has been corrupted. Task Agent C identifies that Task Agent A is capable of performing the grouping by type task and requests that Task Agent A group the selected books by type. Task Agent A performs this activity by using the variables passed to it by Task Agent C as input. Then Task Agent A returns the result of this task execution, which is a new grouping of books by type, to the originating Task Agent C.

Task Agent C now has the list of books selected and arranged by the instructor and grouped by type. It then automatically executes the output task of "View Current (T06)" to display this grouping of books by type to the instructor. The instructor is satisfied with the arranged results and so stops the process.

Chapter 6 Evaluation and Comparison to Related Work

This chapter evaluates the TaMeX intelligent-agent framework by comparing it to related work and by describing the prototypes that were developed using it. TaMeX is compared to its initial version, Web services standardization, and other intelligent integration architectures and languages. Three integrated applications developed using the TaMeX framework are described: book-finding and pharmacy shopping of the comparative-shopping genre and Surgery 446 clinical reporting of the logbook recording genre.

6.1 Comparisons to Related Work

This section evaluates this work on TaMeX by comparing it to the related work in four main areas: the first version of TaMeX, other intelligent integration architectures, other intelligent integration languages and Web services standardization work.

6.1.1 First Version of TaMeX

The TaMeX project is a project to create a task-structure based architecture to integrate existing web-based applications with additional services. The first version of this architecture [SS00, STS00] contributed extensively to the research objective of adapting existing web-applications to communicate within the integration and less extensively to the other research objectives. This thesis, in contrast to the first version, focuses extensively on the other research objectives and less extensively on the research objective concentrated on by the first version. A list of the research objectives, along with the contributions made by the first version of the TaMeX project and the further contributions made by this thesis, follows.

1. ***Web-based Application Adaptation*** - the development of intelligent methods for adapting existing web-based applications so that they "speak" the same language, in terms of both syntax and semantics.

   Contribution of the First Version - The first version of this architecture focused mainly on this research objective. A wrapper-construction process was developed that allows a (semi-) automatic adaptation of existing web-based applications from their original HTML-based user interfaces into a common, domain-specific vocabulary. This wrapper-construction process, described previously in Section 4.4, is essentially an example-based learning process. This process exploits the tree-structure of HTML documents to identify rules for extracting the information of interest from the HTTP requests accepted by the web applications and their HTML responses.

Contribution of this Thesis - This thesis did not contribute anything further to the development of the actual wrapper-learning algorithm itself. However, it cleaned up its output, the wrapper specification files, by converting them to a format that conforms to the XML syntax. See the discussion of this in the *Expressive Language* point that follows. In addition, it demonstrated the use of this wrapper-construction process in the development of three TaMeX integrated application prototypes that are discussed in Section 6.2.

2. ***Expressive Language*** - the development of a language with well-defined semantics for specifying the application domain, the pre-existing services, the supported services and their composition.

   Contribution of the First Version- The first version of this architecture provided two simple languages to specify the pre-existing services and the supported services and their composition. It did not provide a language for specifying the application domain.

   - *Application Domain* - There was no language provided for specifying the application domain. Every model of the application domain was created from scratch using an XML Schema.
   - *Pre-existing Services* - The wrapper-specification language was provided to specify the protocol for accessing the web-based application services and the grammar for extracting the information provided by these services. This language was specified using an XML Document Type Definition (DTD) [XML]. A DTD is a schema notation that was a precursor to the XML Schema Definition (XSD)

[XSD].

- *Supported Services and their Composition* - The mediator task-structure specification language and the associated presentation stylesheet was provided to specify the services available and their composition. This language was specified using an XML Document Type Definition (DTD) [XML].

Contribution of this Thesis - This thesis provided languages, with well-defined semantics, to specify all the areas cited in this research objective: the application domain, the pre-existing services, the supported services and their service composition. Additionally, it provided a language to specify user customizations of the task structure model according to individual user preferences at run-time. The semantics of these languages are defined by the constraints that are described in Section 2.2.3.

- *Application Domain* - The Domain Model of the TaMeX integration-specification language, described in Section 2.2.1, is provided for specifying the application domain.
- *Pre-existing Services* - This thesis converted the wrapper specification files, which were provided by the first version of TaMeX, to a format that conforms to the XML syntax. Originally, these files were specified using an XML Document Type Definition (DTD). This is now an outdated technology. As well, there is no way of parsing DTDs in the same way as XML files can be parsed. To clean up these files and have a consistent technology throughout the TaMeX project, these files were converted from DTDs to XML files.
- *Supported Services and their Composition* - The Task Model of the TaMeX integration-specification language, described in Section 2.2.2, is provided for specifying the supported services and their service composition. This Task Model is essentially a new creation since only the foundational concepts for the development of this Task Model came from the task-structure specification language developed in the first version of TaMeX. As well, the task-structure language developed in the first version could not be directly built upon because it was specified using the outdated technology of a DTD for the schema notation.

The role of a schema is to define which documents are valid and what constraints they must meet for them to be meaningful. DTDs, as a schema notation, can specify some limited control over the structure of their elements, such as which elements are contained within which other elements. . However, they cannot specify anything about the text contained within these elements. They also offer only limited control over the specification of attributes, as there is not even a mechanism for specifying that an attribute must be numeric. The XML Schema Definition (XSD), standardized by the W3C, addressed these limitations of DTDs. The XSD is a schema notation that includes the structural and control rules for a document, as well as detailed rules for defining data types. The Task Model Schema was specified using an XSD and incorporating the few, simple concepts from the task-structure DTD of the first version.

The language provided from the first version provided the concepts that a task model is composed of tasks. Additionally, each task can be composed of other tasks (its subtasks) and has input, output, and an operator. This thesis extended these basic concepts to include detailed specifications of each type of task and another level of composition within each task. The detailed specifications of all tasks include their conceptual constraints, functional constraints, and other agents that are capable of performing them. The new level of composition of a task, called the task statement, defines the allowable constituents of each task type. For the grouping task, the control interdependencies and execution style of their subtasks are specified within the task model. The min and max attributes are used to specify the minimum number of subtasks that must be executed before a task is considered complete and the maximum number of subtasks that may be executed for a particular task, respectively. Defining these task details in the declarative specification allows the TaMeX task agent to interpret the task structure generically without resorting to a hard-coded process.

Additionally, constructs were added to allow the specification of the user profile information. The *autogen* task statement is used to specify the information necessary for creating the input form for the default values for this task to be used with the user profiles. This information is then used by the TaMeX task agent, at run-time, to automatically generate the input form for this task in the *Messages/Enter Default Information* column of the *Profile Options* screen.

- *User Customizations* - The Profile Model of the TaMeX integration-specification language, described in Section 2.2.4, is provided for specifying the customization of the task structure model according to individual user preferences.

3. ***Flexible Execution Run-time Environment*** - the development of a flexible and robust run-time environment for executing the integrated applications and delivering the desired overall service.

   Contribution of the First Version - The first version of this architecture provided a single-user environment that had some components that were generic and reusable across similar applications. The central component, the mediator, based its processing on its task structure and was the intermediary between the user's browser and the constructed wrappers that accessed the web-based application services. The task structure was used to specify the flow of information within the integrated application, the internal information processing of the mediator, and the menu of tasks from which the user interacts with the mediator. This version of the mediator used a hard-coded process to interpret its non-deterministic task structure at run-time to dynamically combine the wrapped applications. The task structure of the mediator allowed newly developed components to be integrated easily with the existing functionalities.

   The wrapper-learning algorithm and the wrapper-construction process provided are reusable for any web-based application in any domain. The wrapper driver component, which uses the wrapper protocol and response grammar specifications created in the wrapper-construction process, is also generic. It uses these specifications to execute the existing web-based application services and translate between the integrated application's domain model and the individual domain models of the wrapped applications.

   Contribution of this Thesis - The first version of the TaMeX architecture could not be considered an application framework [BS98, FHLS97] although it had a few components that were generic and reusable across similar applications. This thesis extended this simple, single-user, single mediator environment developed in the first version of TaMeX into a framework that is multi-user, multi-agent, flexible, and robust. It also contributed a declarative specification model, the Main Configuration Model described in Section 2.3.2, for configuring the system for an individual task agent. This model includes a registry for the wrappers of the web-based application services, a feature that was not included in the first version of TaMeX. This feature is important because it increases the extendibility of the TaMeX integrated application by allowing new web-based application services to be easily added to the system.

4. ***User Customization Methods*** - the development of user customization methods for composing the services at run-time.

   Contribution of the First Version - The first version of this architecture did not address this research objective of user customization methods at all.

   Contribution of this Thesis - This thesis provided user customization methods as described in Sections 2.2.4 and 3.2.

5. ***Design-time Toolkit*** - the development of tools to support the development of integrated applications.

   Contribution of the First Version - The first version of this architecture contributed a wrapper-construction toolkit that allows a developer to semi-automatically construct the wrapper specification files.

   Contribution of this Thesis - This thesis contributed templates and schemas for creating declarative models and a set of validating tools for these models.

6.1.2 Intelligent Integration Architectures

Information Agents - A research effort, similar in objectives and scope to TaMeX, is the "Information Agents" project [KMAAMPT01], which has also produced an example-based wrapper-construction process [MMK01] with supporting tools, a planning language for composing wrappers [ABKMOM02, BDKM00], and a run-time execution environment. At design time, a developer creates a separate wrapper, as a little relational database, from each web page to be used in the integrated application. This wrapper-construction process is facilitated by the use of modeling tools. At run-time, the planning language decomposes the end user's query into simpler queries, each of which can be answered using a single wrapper. Then, the planning language creates the final answer to the user's query by composing all the responses from the wrappers.

Comparison with TaMeX - The fundamental difference between these two projects lies in the way they view the underlying services and their composition. The "Information Agents" project views web applications as databases; consequently, its wrappers correspond to queries and their composition to query planning. TaMeX adopts a more general approach to the integration problem. It views web applications as providing services and its composition language distinguishes among different types of information-transformation and user-interaction tasks. This approach enables a wider range of types of compositions, requiring the maintenance of complex state information and including user interaction, wrapped service accesses, and invocation of new functional components. Finally, the main advantage of the TaMeX approach is its use of reflection to monitor the run-time execution of the integrated application and its support for distributed collaboration among its agents for addressing run-time failures.

RETSINA - Another related agent-based integration framework, developed at Carnegie Mellon University, is RETSINA (Reusable Environment for Task Structured Intelligent Network Agents) [PKPSS99, PSS00, SP98]. The RETSINA multi-agent architecture is used to develop a distributed collection of agents that cooperate asynchronously to gather, filter, and integrate information to support a user's problem-solving needs. The agents in the system actively seek out information and collaborate with each other and their users. Information gathering is integrated with problem solving and decision support.

There are four agent types in RETSINA: interface agents, task agents, information agents, and middle agents. Of these types, the first three types are reusable and can be adapted by the developer to address his/her domain-specific needs. The architecture for each agent consists of several reusable modules:

- Communication and Coordination module - handles the communication (messages and requests) from other agents
- Planning module - creates a plan (task structure) that satisfies the input goals
- Scheduling module - schedules (orders) the tasks from the planning module's plan
- Execution module - monitors the execution of the tasks

Comparison with TaMeX - RETSINA's agents employ a plan-based description of their capabilities, similar to the task structure specifications of the TaMeX agents. RETSINA tasks are represented using the Hierarchical Task Network (HTN) formalism [EHN94], which decomposes a high-level task into a tree structure of subtasks, including indicating how the outcome of one task is propagated to the other tasks. RETSINA tasks are either primitive tasks, which describe actions that the agent can perform directly, or complex tasks, which describe the composition of other primitive and complex tasks. RETSINA's primitive and complex tasks are equivalent to TaMeX leaf and grouping tasks respectively. Complex tasks are used by the RETSINA agent's scheduling module to control the execution of primitive tasks, similarly to how TaMeX agents employ the grouping tasks. Both types of agents monitor the process of executing their plans, ensure that constraints are satisfied and take action upon failure, including constraint failure. Though both projects are similar in their utilization of a task structure based plan, the initial creation of these plans is different. RETSINA agents construct their own plans using an internal planning module whereas TaMeX agents employ pre-specified plans that are selected by the user from a set of provided task models.

An important difference between the two frameworks is the underlying implementation platform: the syntax of the TaMeX integration-specification language is based on XML -as opposed to the RETSINA specifications that use a proprietary format - thus making the deployment of TaMeX applications on the Web quite straightforward.

6.1.3 Intelligent Integration Languages

Related to the TaMeX integration-specification language, Elio, Haddadi, and Singh [EH99, EHS00] proposed high-level goal-specific structures to express the semantics of the discourse between a human and an agent, or two agents, who cooperate to accomplish a common goal, such as a successive refinement search task. Their research focused on conversation policies and used jointly shared abstract task specifications to determine the agents' intentions; thus constraining the messages required to communicate with each other by constraining their intentions.

The abstract task specifications were based on a state-space formulation for a problem that requires a goal test that indicates whether a state has solved the problem, an initial state specification, and the specification of a set of task actions that are executed on states to transform them from one state to the next. Task actions have checkable preconditions and post-conditions associated with them, as well as a reference to which agents are capable of fulfilling them. Preconditions must be satisfied by the current state before the task action can be executed and post-conditions define the new state created by the execution of the task action.

An agent's intentions were separated into and modeled as two distinct categories: task intentions and discourse intentions. The task intentions correspond to the full set of task actions that either agent can take for the advancement of a task-related goal. The discourse intentions correspond to the full set of precondition- and post-condition-satisfying actions that either agent can take for the advancement of information exchange that supports a task intention.

The protocol for exchanging information between two agents about a task action was defined as a set of input parameters, including a given state, and a set of output parameters, which fully define the state resulting from this task action.

Comparison with TaMeX - The TaMeX integration-specification language with its task models corresponds well to the abstract task specifications of these researchers. Both research works model their task specifications on a state-space formulation for a problem with inspectable preconditions and post-conditions and specify agent functionality.

The main difference between the two is that the TaMeX project's focus is not on agent communication and therefore does not model discourse intentions. In TaMeX, an agent only collaborates with another agent when it is incapable of performing a task or upon failure of performing a task. As well, a TaMeX agent only collaborates with other agents that are in its agent registry and by definition a registered agent is always considered ready, willing, and able to perform the tasks for which it is registered. Therefore, there are no lengthy dialogues between agents. The only message that is sent to a backup agent is a request to execute a task statement and the only message returned from the backup agent is the result message from the execution.

When a TaMeX agent, the orchestrating agent, is unable to successfully complete a task for whatever reason, including a precondition not being met, it looks up the other agents that are capable of performing this task. Rather than initiating a lengthy discourse with these agents, the orchestrating agent successively sends a request to each agent on the list of backup agents until a successful response is received from one of these agents. This request contains the task statement to be executed and a list of current variables. If this request fails, the next backup agent is tried. If the request succeeds, the backup agent returns a response to the orchestrating agent that includes any changes to the list of variables, the output produced, and any execution exceptions. The orchestrating agent then checks this returned response and if successful it updates its local information and continues normal processing. Otherwise, the next backup agent on the list is tried.

6.1.4 Web Services

The TaMeX view of existing web applications as "providing services" and its use of XML and related languages match well with recent efforts on the Web services stack of standards [Web] based on the Extensible Markup Language, XML [XML]. Web services are a standardized way of integrating Web-based applications, which constitute a set of resources intended for reuse and interoperation, over standard Internet protocols. The architecture for Web services is based on middleware design principles for communication between applications. Thus, the Web services architecture (WSA) is a "Web" version of the service-oriented architecture (SOA), which is the architecture for connecting service consumers with service providers via the assistance of service brokers. The three core functional architecture components of a SOA needed to perform its operations are the transport component, the description component, and the discovery component. These components specify how the services to be offered by the service provider are described (description component) and then registered (discovery component) with a service broker for later discovery (discovery component) and invocation (transport component) by the service consumer.

Early Web services standardization activities have focused on standardizing the protocols for the Web-based SOA core functional architecture components. Though not yet formal standards, the standards for these basic Web service components - transport (SOAP), description (WSDL) and discovery (UDDI) - are stable and are the de facto standard [See03]. SOAP, the Simple Object Access Protocol, specifies a standard communications protocol for Web services that allows one application to send an XML message to another application [SOAP]. WSDL, the Web Services Description Language, specifies a standard way, expressed in XML, to describe a Web service [WSDL]. UDDI, Universal Description, Discovery and Integration, provides a standard way to register and find Web services [UDDI].

Since it is likely that a particular Web service will be used in concert with other services, it is necessary to have a method of composing these services to model the actual business processes. The standards for this Web services orchestration are still under development - several XML grammars have been proposed, including the strong contender of IBM's Web Services Flow Language (WSFL) [Ude02]. WSFL is an XML language for describing the compositions of Web services [Ley01].

Efforts are also under way, by the Web Services Interactive Applications Technical Committee (WSIA TC), to provide XML vocabularies and Web services interfaces to allow end users to interact with Web services. This committee is supporting the adoption of IBM's Web Services Experience Language (WSXL) for standardizing this interaction with the user. WSXL services provide operations for managing the life cycle of the interactive Web application, both accepting user input and producing output presentation markup. WSXL services will also allow Web applications to be available to end users through a variety of channels, including a browser, indirectly through a portal, or embedded into a third party Web application [WSXL].

Comparison with TaMeX - The basic Web services components (WSDL, UDDI, SOAP) with the addition of the WSFL and WSXL components correspond well with TaMeX components.

First, the TaMeX component correspondence with the basic Web services components will be considered. WSDL defines how Web services should be specified using XML. The TaMeX wrapper specification of existing web applications specifies the adaptation of existing HTML/HTTP-based APIs into XML. Thus, it essentially amounts to the reengineering of a legacy web protocol into a syntax similar to WSDL.

The UDDI registry contains a catalog of WSDL specifications of Web services. UDDI provides the means to publish these specifications to the registry and then later discover them for reuse by other applications. TaMeX contains a similar type of registry, with the exception being that in TaMeX the registry is used internally by the task agent rather than being available to outside applications. The TaMeX registry is the Wrapper Registry of the Main Configuration Model, described in Section 2.3.2, which is a catalog of the wrappers available to a particular task agent.

SOAP provides a messaging framework that allows applications to communicate via XML messages. The run-time TaMeX environment does not use SOAP however; instead it uses special components implemented as Java servlets to execute the wrappers and the integration workflow with other task agents. These components communicate using Java Remote Method Invocation (RMI) [JavaRMI].

WSFL and WSXL work together to allow the user to interact with the Web application. WSFL provides the flow model, which specifies the information and control flow among Web services, and WSXL provides the user interface - accepting user input and producing the presentation of the output. The domain and task models of the TaMeX integration-specification language together correspond approximately to WSFL; both are aimed at specifying the information and control flow among components. Furthermore, the user-interaction specification aspect of the language attempts to address the issues with which WSXL is concerned.

6.2 Case Studies

We have developed two comparative-shopping integrated applications with TaMeX in two different domains, i.e., book finding and pharmacy shopping. Each of these applications integrates several existing web applications and performs a different variant of a comparative, exploratory search of the products of these applications.

Although we have focused on comparative-shopping as an interesting family of applications and a desktop computer as the implementation platform, TaMeX can be used to develop integrated applications in other areas, such as logbook recording, and on other implementation platforms, such as mobile devices. To illustrate this, we developed a medical student logbook for surgical interns on rotation who need to keep track of the operations they attend as part of their surgery rotation. In addition to integrating existing web applications that are queried for their information, its workflow also includes several types of transactions, one for each type of surgical experience they may have during their rotation. This application was implemented with two variants for different clients. As well as being able to run on a desktop, it can also be run on a mobile device, such as a palm pilot.

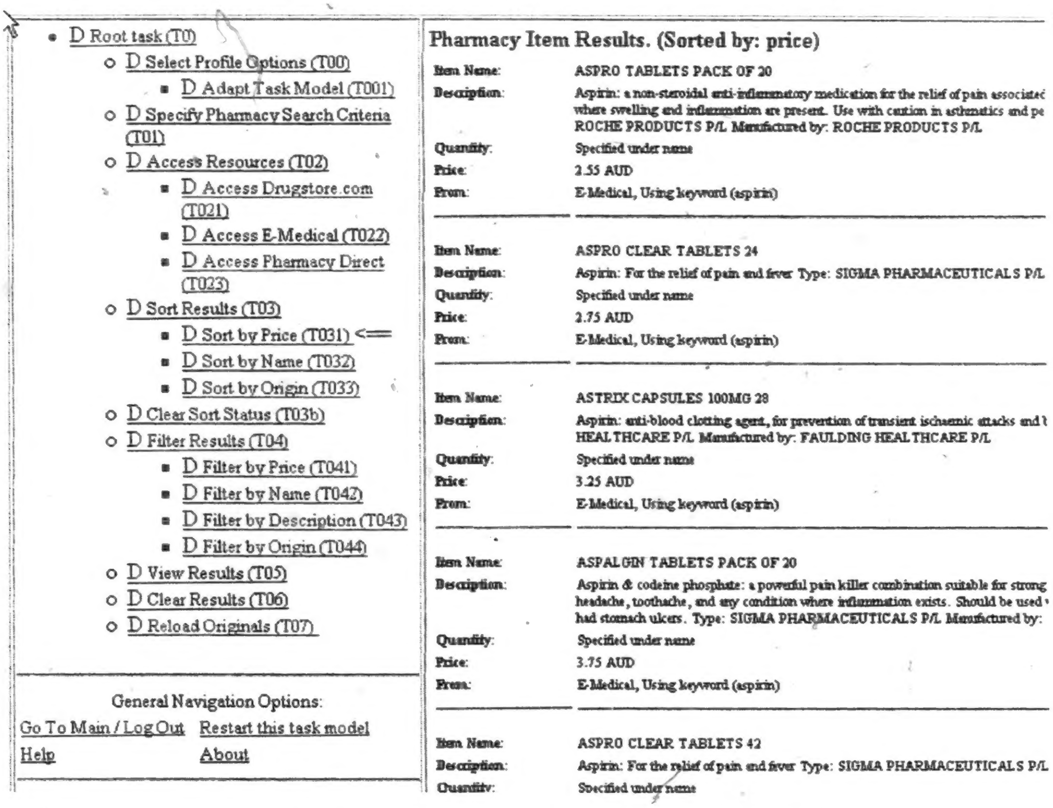
6.2.1 Book-Finding Prototype (Comparative-shopping)

The *Book-Finding* prototype is an example of a comparative-shopping application, in the book-finding domain, that integrates three book-selling web applications with sorting, grouping, and filtering services. This application is used as an example throughout this thesis. It is described in Section 2.4 and its development and run-time behavior are described in detail in Chapters 4 and 5 respectively.

6.2.2 Pharmacy Shopping Prototype (Comparative-shopping)

The *Pharmacy Shopping* prototype is an example of a comparative-shopping application, in the pharmacy shopping domain, that integrates three pharmaceutical supply web applications with sorting and filtering services. See Figure 34 for the task interaction screen returned after sorting the results returned from a request for 'aspirin' from the emedical online pharmacy (http://www.emedical.com.au).

Although this application is in a different domain from the Book-finding application, it belongs to the same family of related applications, namely comparative-shopping. This means the exact same development steps as used for the Book-finding application were followed including using the same "comparative-shopping" Domain Model Template to create the Pharmacy Shopping Domain Model Schema.



- D Root task (T0)
  - D Select Profile Options (T00)
    - D Adapt Task Model (T001)
  - D Specify Pharmacy Search Criteria (T01)
  - D Access Resources (T02)
    - D Access Drugstore.com (T021)
    - D Access E-Medical (T022)
    - D Access Pharmacy Direct (T023)
  - D Sort Results (T03)
    - D Sort by Price (T031) <==
    - D Sort by Name (T032)
    - D Sort by Origin (T033)
  - D Clear Sort Status (T03b)
  - D Filter Results (T04)
    - D Filter by Price (T041)
    - D Filter by Name (T042)
    - D Filter by Description (T043)
    - D Filter by Origin (T044)
  - D View Results (T05)
  - D Clear Results (T06)
  - D Reload Originals (T07)

General Navigation Options:

Go To Main / Log Out	Restart this task model
Help	About

Pharmacy Item Results. (Sorted by: price)

Item Name:	ASPRO TABLETS PACK OF 20
Description:	Aspirin: a non-steroidal anti-inflammatory medication for the relief of pain associate where swelling and inflammation are present. Use with caution in asthmatics and pe ROCHE PRODUCTS P/L Manufactured by: ROCHE PRODUCTS P/L
Quantity:	Specified under name
Price:	2.55 AUD
From:	E-Medical, Using keyword (aspirin)

Item Name:	ASPRO CLEAR TABLETS 24
Description:	Aspirin: For the relief of pain and fever Type: SIGMA PHARMACEUTICALS P/L
Quantity:	Specified under name
Price:	2.75 AUD
From:	E-Medical, Using keyword (aspirin)

Item Name:	ASTRIX CAPSULES 100MG 28
Description:	Aspirin: anti-blood clotting agent, for prevention of transient ischaemic attacks and HEALTHCARE P/L Manufactured by: FAULDING HEALTHCARE P/L
Quantity:	Specified under name
Price:	3.25 AUD
From:	E-Medical, Using keyword (aspirin)

Item Name:	ASPALGIN TABLETS PACK OF 20
Description:	Aspirin & codeine phosphate: a powerful pain killer combination suitable for strong headache, toothache, and any condition where inflammation exists. Should be used had stomach ulcers. Type: SIGMA PHARMACEUTICALS P/L Manufactured by:
Quantity:	Specified under name
Price:	3.75 AUD
From:	E-Medical, Using keyword (aspirin)

Item Name:	ASPRO CLEAR TABLETS 42
Description:	Aspirin: For the relief of pain and fever Type: SIGMA PHARMACEUTICALS P/L
Quantity:	Specified under name

Figure 34: Pharmacy Shopping Results Sorted by Price

6.2.3 Surgery 446 Clinical Reporting Prototype (Logbook Recording)

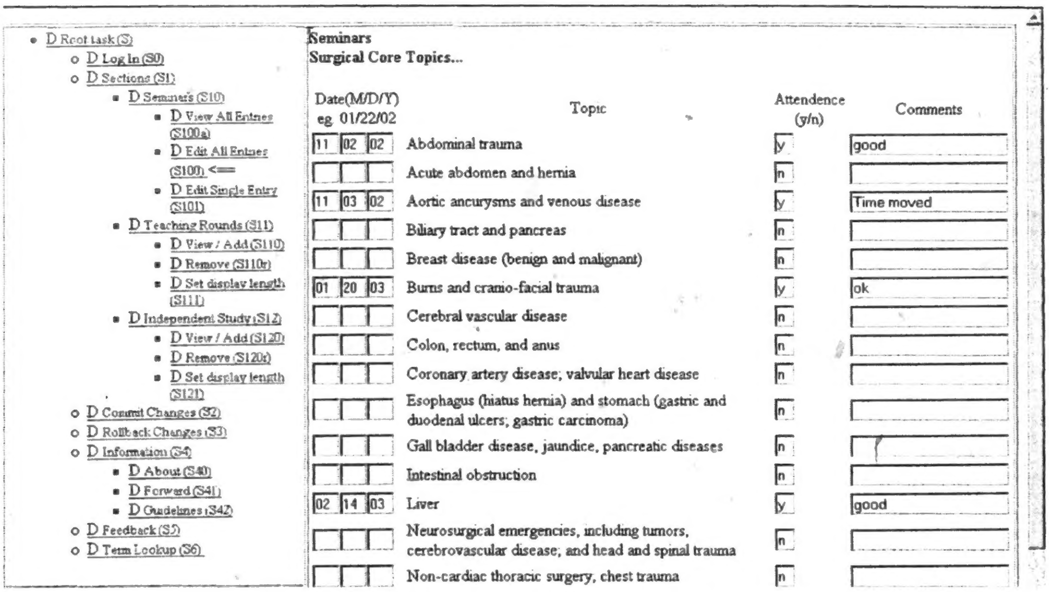
The *Surgery 446 Clinical Reporting* prototype is an example of a logbook recording application, in the medical instruction domain, that integrates one medical terminology web application with new recording and informational services. The services developed were all those required to duplicate the informational and recording capabilities of the *Report on Clinical Activities During Surgery 446* logbook. This prototype is different from the previously described prototypes, not only in that it has a different family of related applications, i.e. logbook recording rather than comparative-shopping, but also in that it can be run on two different platforms: a mobile device such as a palm pilot and a desktop computer.

The medical students taking the course *Surgery 446* of the Department of Surgery at the University of Alberta must record all their clinical activities, related to this course, in a logbook. Throughout the duration of Surgery 446, they compile a detailed record of their experiences including procedures performed or observed, surgical care seminars attended, and independent study undertaken during the general surgery rotation. The satisfactory completion of this compilation is a mandatory component of Surgery 446 and their reports will be evaluated at the conclusion of the course. Currently they record this information manually in the logbook. If they lose the logbook, their record of all their clinical activities for the course is lost. As well, the logbooks can only be in the hands of one person at a time, i.e. the instructors cannot be evaluating the students' progress at the same time that the students are updating their activities.

The Surgery 446 Clinical Reporting prototype was developed to eliminate these problems with the manual method of recording the clinical activities and to provide electronically-accessible information for research and evaluation purposes. However, the students needed an electronic method of recording their activities that duplicated the ready availability of writing in a logbook.

Though the TaMeX applications can be accessed through a web browser from any device, the limited screen "real estate" of a mobile device made it tedious to interact with the system. Therefore, a "parallel" set of XSLT stylesheets was developed that was tailored to small mobile devices with limited screen space. See Figure 35 for the single screenshot of the desktop version of the *Edit All Entries* of the Seminars activity and see Figure 36 for three screen shots of the mobile device version of the same activity. Thus, the same task structure now has a set of alternative stylesheets for device-specific rendering of its user interface, thus improving the usability of the application. The prototype could now be run efficiently using a web browser on a mobile device as well as on a desktop computer. The medical students could then carry the palm pilot with them as they made their rounds and record their activities on the fly.

As well as developing the tasks necessary to duplicate the recording activities contained in the Surgery 446 logbook, we also wrapped a medical terminology web site, www.medterms.com, to provide a medical term lookup function to the students.

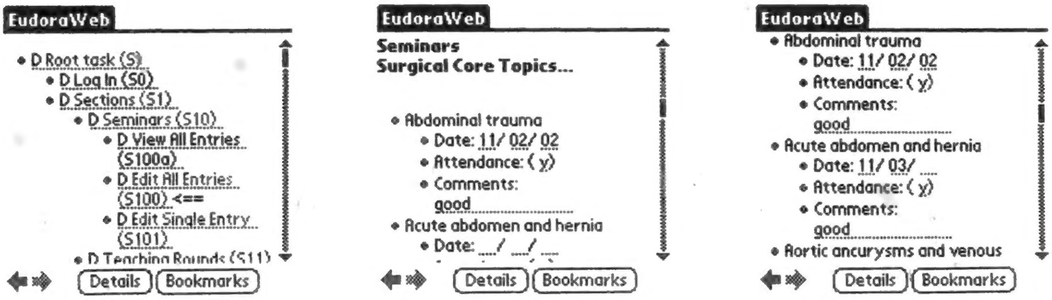


- D Root task (S)
  - D Log In (S0)
  - D Sections (S1)
    - D Seminars (S10)
      - D View All Entries (S100a)
      - D Edit All Entries (S100) <==
      - D Edit Single Entry (S101)
    - D Teaching Rounds (S11)
      - D View / Add (S110)
      - D Remove (S110r)
      - D Set display length (S111)
    - D Independent Study (S12)
      - D View / Add (S120)
      - D Remove (S120r)
      - D Set display length (S121)
  - D Commit Changes (S2)
  - D Rollback Changes (S3)
  - D Information (S4)
    - D About (S40)
    - D Forward (S41)
    - D Guidelines (S42)
  - D Feedback (S5)
  - D Term Lookup (S6)

Seminars
Surgical Core Topics...

Date(M/D/Y) eg 01/22/02	Topic	Attendence (y/n)	Comments
11 02 02	Abdominal trauma	y	good
	Acute abdomen and hernia	n	
11 03 02	Aortic ancurysms and venous disease	y	Time moved
	Biliary tract and pancreas	n	
	Breast disease (benign and malignant)	n	
01 20 03	Burns and cranio-facial trauma	y	ok
	Cerebral vascular disease	n	
	Colon, rectum, and anus	n	
	Coronary artery disease; valvular heart disease	n	
	Esophagus (hiatus hernia) and stomach (gastric and duodenal ulcers; gastric carcinoma)	n	
	Gall bladder disease, jaundice, pancreatic diseases	n	
	Intestinal obstruction	n	
02 14 03	Liver	y	good
	Neurosurgical emergencies, including tumors, cerebrovascular disease; and head and spinal trauma	n	
	Non-cardiac thoracic surgery, chest trauma	n	

Figure 35: Surgery 446 Desktop Version of Edit All Entries (S100) of the Seminars



- D Root task (S)
  - D Log In (S0)
  - D Sections (S1)
    - D Seminars (S10)
      - D View All Entries (S100a)
      - D Edit All Entries (S100) <==
      - D Edit Single Entry (S101)
    - D Teaching Rounds (S11)

Seminars
Surgical Core Topics...

- Abdominal trauma
  - Date: 11/ 02/ 02
  - Attendance: (y)
  - Comments: good
- Acute abdomen and hernia
  - Date: __/ __/ __

- Abdominal trauma
  - Date: 11/ 02/ 02
  - Attendance: (y)
  - Comments: good
- Acute abdomen and hernia
  - Date: 11/ 03/ __
  - Attendance: (y)
  - Comments: good
- Aortic ancurysms and venous

Figure 36: Surgery 446 Mobile Device Version of Edit All Entries (S100)

Chapter 7 Conclusions

In this thesis, we discussed TaMeX, a software framework for developing intelligent multi-agent applications for integrating web-based application services. This architecture is based on an extensible integration-specification language and a run-time environment consisting of reflective intelligent agents able to interpret and execute integration specifications, defined in the language.

7.1 Contributions

The contributions of this thesis to the TaMeX project built on the work contributed by the first version of TaMeX, as detailed in Section 6.1.1. The main contributions of this thesis to the work on the TaMeX intelligent-agent framework are

1. its extensible integration-specification language,
2. its flexible, adaptive, distributed intelligent-agent environment for run-time execution,
3. its methods for user-based, run-time customization of the integration specifications, and
4. its design-time toolkit.

As well, three prototypes, as described in Section 6.2, were developed for the demonstration of the TaMeX intelligent-agent framework. Two of these prototypes, book finding and pharmacy shopping, were of the comparative-shopping genre and used the comparative-shopping Domain Model Template to create their Domain Model Schemas. The third prototype, Surgery 446 clinical reporting, was a logbook recording application that demonstrated the use of a "parallel" set of XSLT stylesheets for device-specific rendering of the user interface.

7.1.1 Integration-Specification Language

The integration-specification language is a language, with well-defined semantics, for declaratively modeling the various aspects of the integrated application. This language includes primitives for representing the concepts of the application domain and their relations, the different types of information-exchange tasks that can be accomplished in the domain either through the wrapped original web-based applications or through their composition, the semantic constraints applicable to the tasks and the domain concepts and the user preferences regarding task composition and domain concept values. The integration-specification language is based on XML and its related technologies and is well aligned to the Web services stack of standards. The use of XML with its declarative modeling provides an easily extensible implementation platform for TaMeX applications.

7.1.2 Intelligent-agent Run-time Execution Environment

The run-time execution environment is a multi-user, multi-agent, flexible and robust run-time environment for executing integrated applications. The intelligent task agent is the key component of this run-time environment. Web application service compositions specified in the TaMeX integration-specification language are executed at run-time by its intelligent task agents. They are responsible for dynamically adapting these compositions according to the user preferences. Furthermore, the task agents interpret these customized task structures to interact with the end user and the underlying web applications to accomplish the user's tasks. Finally, the reflection capabilities of the intelligent task agents enable them to recognize failures, in response to which they may collaborate with each other to recover and complete the user's task.

7.1.3 User-based Run-time Customization

The various options available for run-time customization of the non-deterministic task-structure model are specified by the developer using the profile model integration-specification language. At run-time, the end users employ the tasks provided for customization, specifically the *Select Profile Options* task and the *Adapt Task Model* task, to adapt the task-structure model according to their own individual preferences.

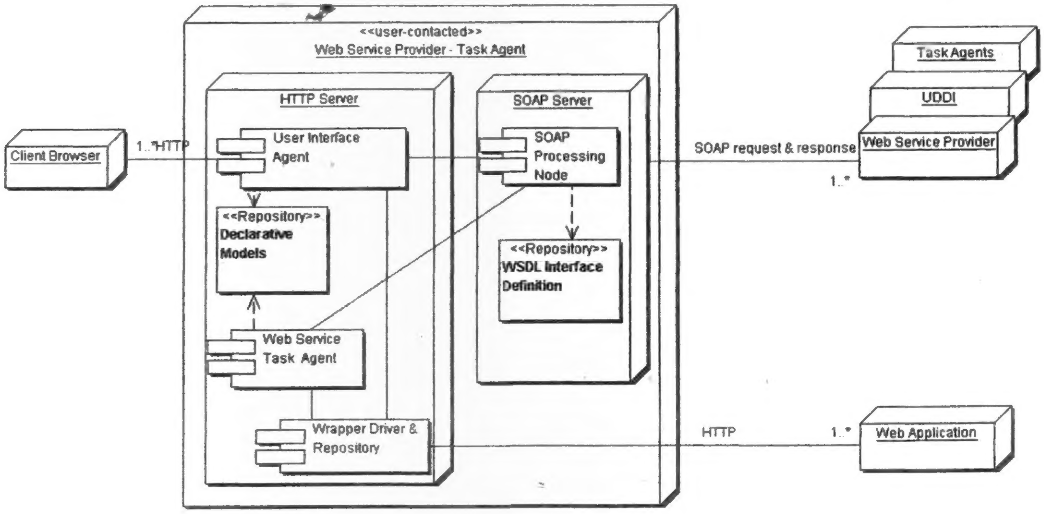
7.1.4 Design-time Toolkit

This thesis contributes templates and schemas that allow a developer to create declarative models in the integration-specification language and a corresponding set of validating tools to ensure that the developed models conform to the rules. The validation of the models is discussed in Section 4.6. These contributions, combined with the wrapper construction toolkit provided by the first version of TaMeX, support the development of integrated applications.

7.2 Future Work

There are three main areas of consideration for future work: integrating TaMeX with Web services standards, providing support for the composition of the task structures used in TaMeX, and providing a look and feel to the users' interaction with TaMeX that is similar to the experiences they had while interacting with the original Web sites.

7.2.1 Web Services Standards Integration



<<user-contacted>>
Web Service Provider - Task Agent
HTTP Server
SOAP Server
Client Browser
1..*HTTP
User Interface Agent
SOAP Processing Node
<<Repository>>
Declarative Models
<<Repository>>
WSDL Interface Definition
Web Service Task Agent
Wrapper Driver & Repository
SOAP request & response
1..*
Task Agents
UDDI
Web Service Provider
HTTP
1..*
Web Application

Figure 37: Web Services Component Diagram for a Task Agent

Since TaMeX is XML based and, as discussed in Section 6.1.4, corresponds well to Web services standards the next step is to integrate TaMeX with these standards. In this discussion, we will consider creating TaMeX task agents as Web services that can access other services as Web services or via the current wrapper method and interact with the users the present way.

Figure 37 depicts the component diagram for a Web service enabled TaMeX task agent. A task agent now consists of a repository of wrapper specification files, a wrapper driver component, a repository of declarative models used by this agent, two interfaces for accessing this task agent, and a SOAP processing node with its repository of WSDL interface definitions.

The *repository of wrapper specification files*, the *wrapper driver component*, and the *repository of declarative models* are the same as previously described in Section 2.3.1

The components for the *two interfaces for accessing* the task agent are a user interface agent and a Web service agent. The user interface agent provides services to the end users and the Web service agent provides services to other task agents and other applications - i.e. it is a Web service. The user interface agent access is the same as before. However, the Web service agent access replaces the previous subcontractor agent access. Now, a task agent is subcontracted by another task agent (or any other application) via Web services and SOAP messages are used for communication in exactly the same manner as for any other Web service provider.

The *SOAP processing node* with its *repository of WSDL interface definitions* performs the same functions for Web services as the wrapper driver component with its associated repository of wrapper specification files performs for wrapper-encapsulated Web applications. However, as well as accessing these Web services that are equivalent to the current wrapper-encapsulated applications, the SOAP processor could also access any other Web service including the UDDI registry and the other task agents that are used for subcontracting.

7.2.2 Support for Task-Structure Composition

Currently, the process of finding the subtasks required to solve a user's problem is a manual process. The subtasks either have to be developed by the designer or already known to the designer - she cannot reuse existing components that she doesn't know about. Once all the pieces have been discovered or created, she must then assemble them herself into a hierarchical task model. Support for task-structure composition would speed up this process and facilitate reuse of existing services. There are two areas of support to be considered:

1. Service Discovery - locating services that are potential candidates for subtasks needed
2. Service Composition - automated support for putting the found/created subtasks together into a coherent task structure

With the conversion of TaMeX to use Web services standards, UDDI could be used to discover the services available through Web services. These found services could then be employed within TaMeX.

7.2.3 Interaction Experience Integration

The face of the TaMeX application that the user sees is generated from the task structure and the XSLT stylesheets. This standard look that we developed only changes with a different set of XSLT stylesheets that are device-specific, such as those created in the Surgery 446 Clinical Reporting Prototype for use on mobile devices. There is nothing to tie together what the user sees in the TaMeX integrated application and what she saw when interacting with the original web sites, i.e. the now underlying wrapped Web applications. More work needs to be done in this area of integrating the user's interaction experience that she had with the original web sites.

Bibliography

[ABKMOM02] J. L. Ambite, G. Barish, C. A. Knoblock, M. Muslea, J. Oh, and S. Minton, "Getting from Here to There: Interactive Planing and Agent Execution for Optimizing Travel", *Proceedings of the Fourteenth Innovative Applications of Artificial Intelligence Conference (IAAI)*, 862-869, Edmonton, Alberta, Canada, 2002.

[BDKM00] G. Barish, D. DiPasquo, C. A. Knoblock, and S. Minton, "Dataflow Plan Execution for Software Agents", *Proceedings of the Fourth Annual Conference of Autonomous Agents*, 138-139, Barcelona, Spain, May 2000.

[BS98] D. Brugali and K. Sycara, "Agent technology: A new frontier for the development of application frameworks", *Object-Oriented Application Frameworks*, (M.fayad, D. Schmid, R. Johnson eds.), Wiley, 1998.

[Cha89] B. Chandrasekaran, "Task Structures, Knowledge Acquisition and Machine Learning", *Machine Learning*, 4:341-347, 1989.

[DOM] Document Object Model (DOM) Level 2 Specification, available at http://www.w3.org/TR/1999/CR-DOM-Level-2-19991210/

[EH99] R. Elio and A. Haddadi, "On Abstract Tasks and Conversation Policies", In F.Dignum and M. Greaves (eds.), *Issues in Agent Communication*: 301-314, Springer Verlag, Berlin, 1999.

[EHN94] Kutluhan Erol, James Hendler, and Dana S. Nau, "HTN Planning: Complexity and Expressivity", In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, Seattle, 1994.

[EHS00] R. Elio, A. Haddadi, and A. Singh, "Abstract Task Specifications for Conversation Policies", *Proceedings of Autonomous Agents 2000*: 229-230, 2000.

[FHLS97] G. Froehlich, H.J. Hoover, L. Liu and P. Sorenson, "Hooking into Object-Oriented Application Frameworks", In the *Proceedings of the*

1997 International Conference on Software Engineering, Boston, May 1997.

[HS02] M. Hatch and E. Stroulia, "Reflective Collaborative Agents for Complex Service Integration", In the *Proceedings of the "Intelligent Service Integration" Workshop*, in the context of AAAI 2002.

[JavaRMI] Java RMI specification, available at: http://java.sun.com/j2se/1.4.1/docs/guide/rmi/spec/rmiTOC.html

[KMAAMPT01] Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Ion Muslea, Andrew G. Philpot, and Sheila Tejada: The ariadne approach to web-based information integration. In the International Journal on Cooperative Information Systems (IJCIS) Special Issue on Intelligent Information Agents: Theory and Applications, 10(1/2), pp145-169, 2001.

[Ley01] Frank Leymann,"Web Services Flow Language (WSFL 1.0)", available http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, IBM Software Group, May 2001.

[MMK01] I. Muslea, S. Minton, and C. A. Knoblock, "Hierarchical Wrapper Induction for Semistructured Information Sources", *Journal of Autonomous Agents and Multi-Agent Systems*, 4:93-114, 2001.

[NPU98] M. Nodine, B. Perry and A. Unruh, "Experience with the InfoSleuth agent architecture", *AAAI-98 Workshop on Software Tools for Developing Agents*, 1999.

[PKPSS99] M. Paolucci, D. Kalp, A. Pannu, O. Shehory, and K. Sycara, "A Planning Component for RETSINA Agents", *ATA-99*, Orlando, Florida, 1999.

[PSS00] M. Paolucci, O. Shehory, and K. Sycara, "Interleaving Planning and Execution in a Multiagent Team Planning Environment", *Technical Report CMU-RI-TR-00-01*, Robotics Institute, Carnegie Mellon University, 2000.

[Rap97] W. Rapaport, "Implementation is Semantic Interpretation", *Technical Report 97-15*, Buffalo: SUNY Buffalo, Department of Computer Science, 1997.

[SDPWZ96] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng, "Distributed Intelligent Agents", *IEEE Expert*, 11(6): 36-46, 1996.

[See03] Rich Seeley, "Standards fights slow Web services", available at http://www.adtmag.com/article.asp?id=7554, 2003.

[SG96] E. Stroulia and A.K. Goel, "A Model-Based Approach to Blame Assignment: Revising the Reasoning Steps of Problem Solvers", *Proceedings of the 13th Annual Conference on Artificial Intelligence*, pp. 959-965, AAAI Press, 1996.

[SG97] E. Stroulia and A.K. Goel, "Redesigning a Problem Solver's Operators to Improve Solution Quality, Proceedings of the 15th International Joint Conference on Artificial Intelligence, pp. 562-567, 1997.

[SG99] E. Stroulia and A. Goel, "Evaluating PSMs in Evolutionary Design: The Autognostic experiments", *International Journal of Human Computer Studies*, vol. 51, pp. 825-847, 1999.

[Siir03] H. Siirtola's website, "Parallel Coordinate Explorer", Department of Computer and Information Sciences, University of Tampere, Finland, 2003, available at http://www.cs.uta.fi/~hs/pce/

[Situ01] Q. Situ, "TaMeX: A Task-structure Based Mediation Architecture for Integration of Web Applications Using XML", Master's thesis, University of Alberta, 2001.

[SOAP] Simple Object Access Protocol (SOAP) 1.1, W3C Note, May 2000, available at http://www.w3.org/TR/SOAP/

[Spool99] Jared M. Spool, *Web Site Usability: A Designer's Guide*, Morgan Kaufmann Publishers, 1999.

[SP98] Katia P. Sycara, Anandeep Pannu, "The RETSINA Multiagent System: Towards Integrating Planning, Execution and Information Gathering", *Agents 1998*: 350-351, 1998.

[SS00] Q. Situ and E. Stroulia, "Task-Structure Based Mediation: The Travel-Planning Assistant Example", In the *Proceedings of the 13th Canadian Conference on Artificial Intelligence*, 14-17 May 2000, Montréal, Québec, Canada, 400-410, Vol.1822 Lecture Notes in Computer Science, Springer Verlag, 2000.

[STS00] E. Stroulia, J. Thomson, and Q. Situ, "Constructing XML-speaking wrappers for WEB Applications: Towards an Interoperating WEB", *Proceedings of the 7th Working Conference on Reverse Engineering*, Brisbane, Queensland, Australia, pp. 59-68, IEEE Computer Society Press, 23-25 November 2000.

[UDDI] Universal Description, Discovery and Integration of Web Services (UDDI), available at http://www.uddi.org/

[Ude02] Jon Udell, "Orchestrate services", InfoWorld, available at http://www.infoworld.com/article/02/07/05/020708plweborch_1.html, 2002.

[UML] Unified Modeling Language (UML), UML Resource Center, available at http://www.rational.com/uml/

[Web] Web Services Activity, available at http://www.w3.org/2002/ws/

[Wie95] G. Wiederhold, "Mediation in Information Systems", *ACM Computing Surveys*, 27(2):265-267, June 1995.

[Wie97] G. Wiederhold and Michael Genesereth, "The Conceptual Basis for Mediation Services", *IEEE Expert*, 1997.

[WM98] David E. Wilkins and Karen L. Myers, "A Multiagent Planning Architecture", In the *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, 1998.

[WSDL] Web Services Description Language (WSDL) 1.1, W3C Note, March 2001, available at http://www.w3.org/TR/wsdl

[WSXL] (WSXL) Web Service Experience Language Version 2, IBM Note, April 2002, available at http://www-106.ibm.com/developerworks/webservices/library/ws-wsxl/

[XML] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000, available at http://www.w3.org/TR/REC-xml/

[Xpath] XML Path Language (Xpath) Version 1.0 W3C Recommendation 16 November 1999, available at http://www.w3.org/TR/xpath/

[XSD] XML Schema Part 0:Primer, Part 1:Structures and Part 2:Datatypes, W3C Recommendation, 2 May 2001, available at http://www.w3.org/TR/xmlschema-0/, http://www.w3.org/TR/xmlschema-1/, and http://www.w3.org/TR/xmlschema-2/

[XSL] Extensible Stylesheet Language (XSL) Version 1.0 W3C Recommendation 15 October 2001, available at http://www.w3.org/TR/xsl/

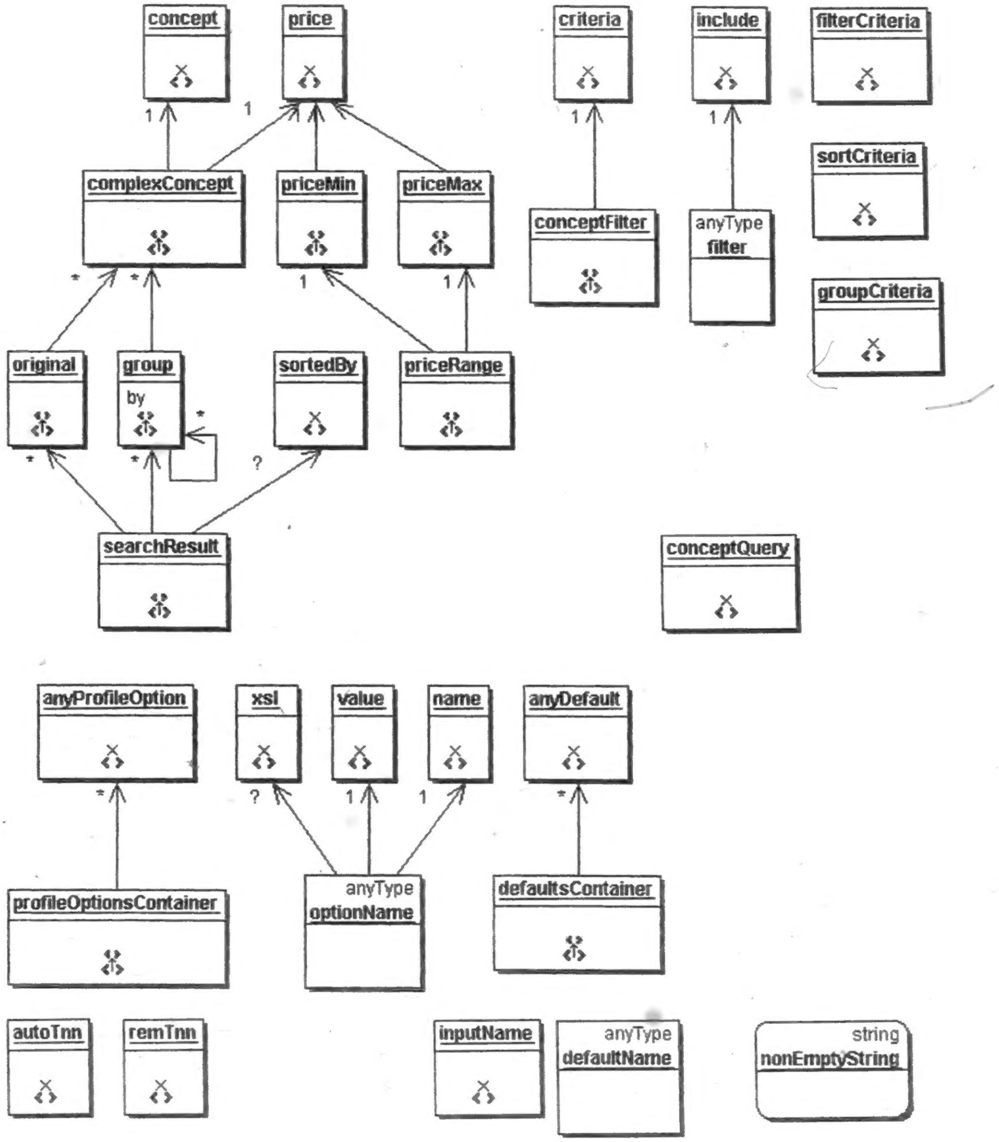
[XSLT] XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999, available at http://www.w3.org/TR/xslt/

Appendices

A - Domain Model (including Profile Model adaptations)

1. Domain Model Template for comparative shopping - XML Structure Diagram

(from TogetherProjects/SchemaFeb6.tpr/SchemaFeb6.tpr.tpr/BookBuyingDomainTemplateNew.xsd)


concept
price
criteria
include
filterCriteria
1
1
1
1
complexConcept
priceMin
priceMax
conceptFilter
anyType
filter
sortCriteria
groupCriteria
*
*
1
1
original
group
by
sortedBy
priceRange
*
*
*
?
searchResult
conceptQuery
anyProfileOption
xsl
value
name
anyDefault
*
?
1
1
*
profileOptionsContainer
anyType
optionName
defaultsContainer
autoTnn
remTnn
inputName
anyType
defaultName
string
nonEmptyString

2. Domain Model Template for comparative shopping - XML Schema

(from tamex/domainModels/DomainModelTemplate.xsd 02/14/03)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd='http://www.w3.org/2000/10/XMLSchema'>

<!--****************************************************************
     DOMAIN CONCEPTS-QUERY:
     -replace the elements with name starting with 'concept' with
      appropriate values for this domain (eg.BookQuery for book finding)
     ****************************************************************-->

 <!-- The query string entered by the user  -->
 <xsd:element name="conceptQuery" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

<!--****************************************************************
     DOMAIN CONCEPTS-PRICE:
     - a generic concept for "comparative shopping" applications
     ****************************************************************-->

 <!-- The price of the item -->
 <xsd:element name="price" type="xsd:string"/>

<!--****************************************************************
     DOMAIN CONCEPTS-MAIN ELEMENT OF INFO for this application:
     - will be a combination of SIMPLE and COMPLEX TYPEs
     - in XML Schema, COMPLEX TYPEs allow elements in their content and
       may carry attributes
     - create as many complex type definitions as are needed using
       a valid construct (see W3C XML Schema Parts 0,1,& 2)
     - ensure that generic field names are replaced with application names
     - update the comments to reflect the concepts being defined
         complexConcept - the object that is retrieved
                  - eg. 'book' in BookFinding app.
         concept - the sub-elements of the object that is retrieved
                 - repeat as many times as necessary
                 - eg. 'title', 'author' in BookFinding app.
     ****************************************************************-->
<!-- a single element of information in the result returned by the server-->
 <xsd:element name="complexConcept">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="concept" minOccurs="1" maxOccurs="1"/>
       <xsd:element ref="price" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

 <!-- definition of the concept  -->
 <xsd:element name="concept" type="xsd:string"/>

<!--****************************************************************
     DOMAIN CONCEPTS-SEARCH RESULT:
     -generic construct for storing the search results of the query
     -replace the following fields with appropriate values for this domain:
       - complexConcept - the object that is retrieved
```

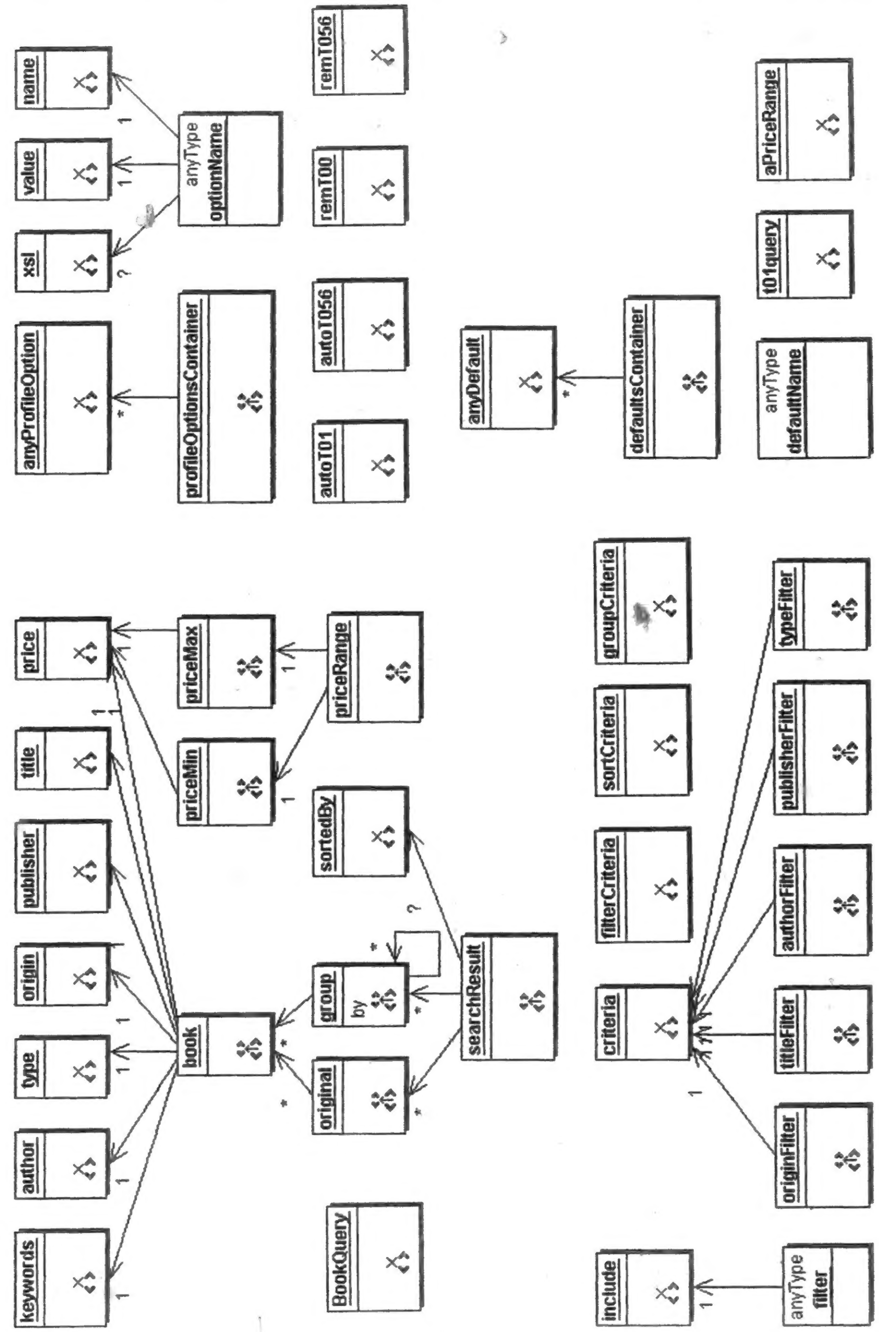
```
               - eg. 'book' in BookFinding app.
****************************************************************-->


<!-- Result returned by the server:
     sortedBy - name of the element that the set is sorted by
     original - the original result returned by the resources
     group - result after it has been sorted, grouped, and/or filtered
 -->
 <xsd:element name="searchResult">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="sortedBy" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
       <xsd:element ref="original" minOccurs="0" maxOccurs="unbounded"/>
       <xsd:element ref="group" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<!-- holds the original result set returned by the resources -->
 <xsd:element name="original">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="complexConcept" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<!-- each group can contain either more groups or objects -->
<!-- the by attribute holds a string describing what the grouping is
     by and this value is optional -->
 <xsd:element name="group">
   <xsd:complexType>
     <xsd:choice>
       <xsd:element ref="complexConcept" minOccurs="0" maxOccurs="unbounded"/>
       <xsd:element ref="group" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:choice>
     <xsd:attribute name="by" type="xsd:string" use="optional" default=""/>
   </xsd:complexType>
 </xsd:element>

<!--****************************************************************
     TASK CONCEPTS-PRICE RANGE for FILTER BY PRICE task:
     - a generic construct
     ****************************************************************-->

<!-- the price range desired for the filter task -->
 <xsd:element name="priceRange">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="priceMin" minOccurs="1" maxOccurs="1"/>
       <xsd:element ref="priceMax" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<!-- the minimum price of the price range element -->
 <xsd:element name="priceMin">
   <xsd:complexType>
     <xsd:sequence>
```

```
       <xsd:element ref="price" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<!-- the maximum price of the price range element -->
 <xsd:element name="priceMax">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="price" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<!--****************************************************************
     TASK CONCEPTS-for GENERAL FILTER tasks:
     - a generic construct
     - replace the following fields with appropriate values for this domain:
       - conceptFilter - the type of concept that is being filtered
     - repeat the 'conceptFilter' construct for all the general filter tasks
     ****************************************************************-->

<!-- Elements used for filtering                                       -->
<!-- the filter criteria for the general filter tasks -->
<xsd:element name="criteria" type="xsd:string"/>

<!-- include filter element -->
<xsd:element name="include" type="xsd:boolean"/>

<!-- specifies the domain concept field that will be used to filter by -->
 <xsd:element name="filterCriteria" type="xsd:string"/>

<!-- general filter type includes 'include' element which flags whether
     or not matching elements should be included or excluded, true
     if they are to be included -->
<xsd:complexType name = "filter">
  <xsd:sequence>
    <xsd:element ref = "include" minOccurs = "1" maxOccurs = "1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name = "conceptFilter">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="filter">
        <xsd:sequence>
          <xsd:element ref = "criteria" minOccurs = "1" maxOccurs = "1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<!--****************************************************************
     TASK CONCEPTS-for SORT tasks:
     - a generic construct
     ****************************************************************-->

<!-- Elements used for sorting                                       -->
 <!-- specifies the domain concept field that will be used to sort by -->
 <xsd:element name="sortCriteria" type="xsd:string"/>
```

```
<!--**************************************************************
     TASK CONCEPTS-for GROUP tasks:
     - a generic construct
     **************************************************************-->

<!-- Elements used for grouping                                   -->
 <!-- specifies the domain concept field that will be used to group by -->
 <xsd:element name="groupCriteria" type="xsd:string"/>

<!--**************************************************************
     TASK CONCEPTS-for PROFILE task:
     - a generic construct

     profileOptionsContainer
     - holds all the options for the profile
     - replace the following fields with appropriate values for this domain:
       - autoTnn - replace 'nn' with the number of the task to be automated
         - repeat the 'autoTnn' construct for all the tasks to be automated
       - remTnn - replace 'nn' with the number of the task to be removed
         - repeat the 'remTnn' construct for all the tasks to be removed

     defaultsContainer
     - holds all the default input information for the profile
     - replace the following fields with appropriate values for this domain:
       - inputName-replace with the input variable name used in the task model
         - repeat the 'inputName' construct for all the tasks with default info
    **************************************************************-->

<!--**************************************************************
     profileOptionsContainer
        - holds all the options for the user profile
**************************************************************-->

 <xsd:element name="profileOptionsContainer">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="anyProfileOption" minOccurs="0"
maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<xsd:element name = "anyProfileOption" abstract="true" type="optionName"/>

 <xsd:complexType name="optionName">
   <xsd:sequence>
     <xsd:element name="xsl" type="nonEmptyString" minOccurs="0"
maxOccurs="1"/>
     <xsd:element ref="name" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="value" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
 </xsd:complexType>

<xsd:element name="autoTnn" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
 <xsd:element name="remTnn" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>

<!-- name and value of a parameter                          -->
      <xsd:element name="name" type="nonEmptyString"/>
      <xsd:element name="value" type="xsd:string"/>

<!--*******************************************************************
     defaultsContainer
        - holds all the default input information for the user profile
*****************************************************************-->

 <xsd:element name="defaultsContainer">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="anyDefault" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<xsd:element name = "anyDefault" abstract="true" type="defaultName"/>

<xsd:complexType name="defaultName">
  <xsd:sequence>
    <xsd:any minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="inputName" substitutionGroup="anyDefault">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="defaultName"/>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<!--*******************************************************************
     BASIC TYPE - nonEmptyString
        - makes sure that if an element is specified then it at
          least has a value, if it does not the java code needs to be
          more complex as empty elements don't have child text nodes and
          this needs to be accounted for.
*****************************************************************-->
<xsd:simpleType name="nonEmptyString">
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="1"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

3. Domain Model Schema for Book Finding - XML Structure Diagram

(from TogetherProjects/SchemaFeb6.tpr/SchemaFeb6.tpr.tpr/BookBuyingDomain030210Rotated.xsd)



name
value
xsl
anyType
optionName
anyProfileOption
profileOptionsContainer
remT056
remT00
autoT056
autoT01
anyDefault
defaultsContainer
aPriceRange
t01query
anyType
defaultName
price
title
publisher
origin
type
author
keywords
book
priceMax
priceMin
priceRange
sortedBy
group
by
original
searchResult
BookQuery
groupCriteria
sortCriteria
filterCriteria
criteria
typeFilter
publisherFilter
authorFilter
titleFilter
originFilter
include
anyType
filter

4. Domain Model Schema for Book Finding - XML Schema

(from tamex/domainModels/BookBuying/BookBuyingDomain.xsd   05/23/03)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd='http://www.w3.org/2000/10/XMLSchema'>

<!--****************************************************************
     DOMAIN CONCEPTS-QUERY:
     -replace the elements with name starting with 'concept' with
      appropriate values for this domain (eg.BookQuery for book finding)
     ****************************************************************-->

 <!-- The query string entered by the user  -->
 <xsd:element name="BookQuery" type="xsd:string" minOccurs="1" maxOccurs="1"/>

<!--****************************************************************
     DOMAIN CONCEPTS-PRICE:
     - a generic concept for "comparative shopping" applications
     ****************************************************************-->

 <!-- The price of the item -->
 <xsd:element name="price" type="priceChecked"/>

<!--****************************************************************
     DOMAIN CONCEPTS-MAIN ELEMENT OF INFO for this application:
     - will be a combination of SIMPLE and COMPLEX TYPEs
     - in XML Schema, COMPLEX TYPEs allow elements in their content and
       may carry attributes
     - create as many complex type definitions as are needed using
       a valid construct (see W3C XML Schema Parts 0,1,& 2)
     - ensure that generic field names are replaced with application names
     - update the comments to reflect the concepts being defined
         complexConcept - the object that is retrieved
                  - eg. 'book' in BookFinding app.
         concept - the sub-elements of the object that is retrieved
                 - repeat as many times as necessary
                 - eg. 'title', 'author' in BookFinding app.
     ****************************************************************-->
<!-- a single element of information in the result returned by the server-->
 <xsd:element name="book">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="title" minOccurs="1" maxOccurs="1"/>
       <xsd:element ref="author" minOccurs="1" maxOccurs="1"/>
       <xsd:element ref="publisher" minOccurs="1" maxOccurs="1"/>
       <xsd:element ref="type" minOccurs="1" maxOccurs="1"/>
       <xsd:element ref="price" minOccurs="1" maxOccurs="1"/>
       <xsd:element ref="origin" minOccurs="1" maxOccurs="1"/>
       <xsd:element ref="keywords" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

 <!-- the title of a book  -->
 <xsd:element name="title" type="xsd:string"/>
<!-- the author of a book  -->
 <xsd:element name="author" type="xsd:string"/>
<!-- the publisher of a book  -->
```

```
 <xsd:element name="publisher" type="xsd:string"/>
<!-- the type of a book: hardcover or paperback  -->
 <xsd:element name="type" type="xsd:string"/>
<!-- the origin of the website that the info came from  -->
 <xsd:element name="origin" type="xsd:string"/>
<!-- the keywords to search for in the book query  -->
 <xsd:element name="keywords" type="xsd:string"/>

<!--***************************************************************
     DOMAIN CONCEPTS-SEARCH RESULT:
     -generic construct for storing the search results of the query
     -replace the following fields with appropriate values for this domain:
       - complexConcept - the object that is retrieved
               - eg. 'book' in BookFinding app.
***************************************************************-->


<!-- Result returned by the server:
     sortedBy - name of the element that the set is sorted by
     original - the original result returned by the resources
     group - result after it has been sorted, grouped, and/or filtered
 -->
 <xsd:element name="searchResult">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="sortedBy" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
       <xsd:element ref="original" minOccurs="0" maxOccurs="unbounded"/>
       <xsd:element ref="group" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<!-- holds the original result set returned by the resources -->
 <xsd:element name="original">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="book" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<!-- each group can contain either more groups or objects -->
<!-- the by attribute holds a string describing what the grouping is
     by and this value is optional -->
 <xsd:element name="group">
   <xsd:complexType>
     <xsd:choice>
       <xsd:element ref="book" minOccurs="0" maxOccurs="unbounded"/>
       <xsd:element ref="group" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:choice>
     <xsd:attribute name="by" type="xsd:string" use="optional" default=""/>
   </xsd:complexType>
 </xsd:element>

<!--***************************************************************
     TASK CONCEPTS-PRICE RANGE for FILTER BY PRICE task:
     - a generic construct
     ***************************************************************-->

<!-- the price range desired for the filter task -->
```

```
 <xsd:element name="priceRange">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="priceMin" minOccurs="1" maxOccurs="1"/>
       <xsd:element ref="priceMax" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<!-- the minimum price of the price range element -->
 <xsd:element name="priceMin">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="price" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<!-- the maximum price of the price range element -->
 <xsd:element name="priceMax">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="price" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<!--*****************************************************************
     TASK CONCEPTS-for GENERAL FILTER tasks:
     - a generic construct
     - replace the following fields with appropriate values for this domain:
        - conceptFilter - the type of concept that is being filtered
     - repeat the 'conceptFilter' construct for all the general filter tasks
     ****************************************************************-->

<!-- Elements used for filtering                                       -->
<!-- the filter criteria for the general filter tasks -->
<xsd:element name="criteria" type="xsd:string"/>

<!-- include filter element -->
<xsd:element name="include" type="xsd:boolean"/>

<!-- specifies the domain concept field that will be used to filter by -->
 <xsd:element name="filterCriteria" type="xsd:string"/>

<!-- general filter type includes 'include' element which flags whether
     or not matching elements should be included or excluded, true
     if they are to be included -->
<xsd:complexType name = "filter">
  <xsd:sequence>
    <xsd:element ref = "include" minOccurs = "1" maxOccurs = "1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name = "typeFilter">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="filter">
        <xsd:sequence>
          <xsd:element ref = "criteria" minOccurs = "1" maxOccurs = "1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "originFilter">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="filter">
        <xsd:sequence>
          <xsd:element ref = "criteria" minOccurs = "1" maxOccurs = "1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "publisherFilter">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="filter">
        <xsd:sequence>
          <xsd:element ref = "criteria" minOccurs = "1" maxOccurs = "1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "authorFilter">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="filter">
        <xsd:sequence>
          <xsd:element ref = "criteria" minOccurs = "1" maxOccurs = "1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "titleFilter">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="filter">
        <xsd:sequence>
          <xsd:element ref = "criteria" minOccurs = "1" maxOccurs = "1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<!--*******************************************************************
     TASK CONCEPTS-for SORT tasks:
     - a generic construct
     *******************************************************************-->

<!-- Elements used for sorting                                       -->
 <!-- specifies the domain concept field that will be used to sort by -->
 <xsd:element name="sortCriteria" type="xsd:string"/>
```

```
<!--****************************************************************
     TASK CONCEPTS-for GROUP tasks:
     - a generic construct
     ****************************************************************-->

<!-- Elements used for grouping                                    -->
 <!-- specifies the domain concept field that will be used to group by -->
 <xsd:element name="groupCriteria" type="xsd:string"/>

<!--****************************************************************
     TASK CONCEPTS-for PROFILE task:
     - a generic construct

     profileOptionsContainer
     - holds all the options for the profile
     - replace the following fields with appropriate values for this domain:
       - autoTnn - replace 'nn' with the number of the task to be automated
         - repeat the 'autoTnn' construct for all the tasks to be automated
       - remTnn - replace 'nn' with the number of the task to be removed
         - repeat the 'remTnn' construct for all the tasks to be removed

     defaultsContainer
     - holds all the default input information for the profile
     - replace the following fields with appropriate values for this domain:
       - inputName-replace with the input variable name used in the task model
         - repeat the 'inputName' construct for all the tasks with default info
    ****************************************************************-->

<!--****************************************************************
     profileOptionsContainer
        - holds all the options for the user profile
****************************************************************-->

 <xsd:element name="profileOptionsContainer">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="anyProfileOption" minOccurs="0"
maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<xsd:element name = "anyProfileOption" abstract="true" type="optionName"/>

 <xsd:complexType name="optionName">
   <xsd:sequence>
     <xsd:element name="xsl" type="nonEmptyString" minOccurs="0"
maxOccurs="1"/>
     <xsd:element ref="name" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="value" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
 </xsd:complexType>

<xsd:element name="autoT01" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
```

```
<xsd:element name="autoT02" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT021" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT022" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT023" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT03" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT031" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT032" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT033" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT034" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT035" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT036" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT03b" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT04" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT041" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT042" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT043" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT044" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="autoT05" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
```

```
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="autoT051" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="autoT052" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="autoT053" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="autoT054" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="autoT055" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="autoT056" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name="remT00" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="remT001" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="remT021" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT022" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT023" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT03" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT031" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT032" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT033" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT034" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT035" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT036" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT03b" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT04" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT041" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT042" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT043" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT044" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT05" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="optionName"/>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
<xsd:element name="remT051" substitutionGroup="anyProfileOption">
   <xsd:complexType>
     <xsd:complexContent>
```

```
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>
<xsd:element name="remT052" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>
<xsd:element name="remT053" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>
<xsd:element name="remT054" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>
<xsd:element name="remT055" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>
<xsd:element name="remT056" substitutionGroup="anyProfileOption">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="optionName"/>
    </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>

<!-- name and value of a parameter                        -->
       <xsd:element name="name" type="nonEmptyString"/>
       <xsd:element name="value" type="xsd:string"/>

<!--*****************************************************************
     defaultsContainer
        - holds all the default input information for the user profile
*****************************************************************-->

 <xsd:element name="defaultsContainer">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="anyDefault" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<xsd:element name = "anyDefault" abstract="true" type="defaultName"/>

<xsd:complexType name="defaultName">
  <xsd:sequence>
```

```
    <xsd:any minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="t01query" substitutionGroup="anyDefault">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="defaultName"/>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="aPriceRange" substitutionGroup="anyDefault">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="defaultName"/>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<!--*******************************************************************
     BASIC TYPE - nonEmptyString
         - makes sure that if an element is specified then it at
           least has a value, if it does not the java code needs to be
           more complex as empty elements don't have child text nodes and
           this needs to be accounted for.
*******************************************************************-->
<xsd:simpleType name="nonEmptyString">
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="1"/>
  </xsd:restriction>
</xsd:simpleType>

<!--*******************************************************************
     PRICE CHECKED TYPE -
         - $ followed by numeric or numeric
*******************************************************************-->
<xsd:simpleType name="priceChecked">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="( )*($)?[0-9 ]*(\.)?[0-9 ]*"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

5. Price Conceptual Constraint-checking Stylesheet

(from tamex/taskModels/myBookBuying/priceCconstraints.xsl 04/07/03)

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml"/>
<xsl:template match="/">

<xsl:for-each select="/Vars/result/searchResult//book">
   <xsl:choose>
      <xsl:when test="boolean(./price)">
         <xsl:if test="starts-with(./price,'$')">
           <xsl:if test="(string(number(substring-after(./price,'$')))='NaN')">
               <xsl:text> The price is not numeric after $  </xsl:text>
               <xsl:value-of select="./price"/>
           </xsl:if>
         </xsl:if>
         <xsl:if test="not(starts-with(./price,'$'))">
            <xsl:if test="(string(number(./price))='NaN')">
               <xsl:text> The price is not numeric  </xsl:text>
               <xsl:value-of select="./price"/>
            </xsl:if>
         </xsl:if>
      </xsl:when>
   </xsl:choose>
</xsl:for-each>

</xsl:template>
</xsl:stylesheet>
```

B - Task Model (including Profile Model adaptations)

1. Task Model Schema - XML Schema

(from tamex/config/TaskModel.xsd   03/31/03)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">


<!--*****************************************************************
     High level description of task model
****************************************************************-->
<xsd:element name = "taskModel">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref = "domain" minOccurs = "1" maxOccurs = "1"/>
      <xsd:element ref = "modelDescription" minOccurs = "0" maxOccurs = "1"/>
      <xsd:element ref = "anyTask" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="nonEmptyString" use="required"/>
    <xsd:attribute name="start" type="nonEmptyString" use="required"/>
  </xsd:complexType>
</xsd:element>

<!--*****************************************************************
     anyTask - an abstract element to be used in a
               substitutionGroup for tasks
****************************************************************-->
<xsd:element name = "anyTask" abstract="true" type="taskType"/>

<!--*****************************************************************
     Individual Task - Grouping Task
        - contains subtask elements and provides structure to
          the task model as viewed by the user
****************************************************************-->
<xsd:element name="groupingTask" substitutionGroup="anyTask">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="taskType">
        <xsd:sequence>
          <xsd:element ref="subtask" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="min" type="xsd:integer" use="optional" default="-
1"/>
        <xsd:attribute name="max" type="xsd:integer" use="optional" default="-
1"/>
        <xsd:attribute name="execute" type="xsd:boolean" use="optional"
default="false"/>
        <xsd:attribute name="execType" type="groupExecutionType" use="optional"
default="sequential"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:simpleType name="groupExecutionType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="sequential"/>
    <xsd:enumeration value="parallel_local"/>
    <xsd:enumeration value="parallel_remote"/>
    <xsd:enumeration value="parallel_hybrid"/>
  </xsd:restriction>
</xsd:simpleType>

<!--*******************************************************************
     Individual Task - Input Task
        - gathers interactive input from the user
*******************************************************************-->
<xsd:element name="inputTask" substitutionGroup="anyTask">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="taskType">
        <xsd:sequence>
          <xsd:element ref = "var" minOccurs = "0" maxOccurs = "unbounded"/>
          <xsd:element ref = "input" minOccurs = "0" maxOccurs = "1"/>
          <xsd:element ref = "output" minOccurs = "1" maxOccurs = "1"/>
          <xsd:choice>
            <xsd:element ref = "xslt" minOccurs = "0" maxOccurs = "1"/>
            <xsd:element ref = "autogen" minOccurs = "0" maxOccurs = "1"/>
          </xsd:choice>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<!--*******************************************************************
     Individual Task - Output Task
        - presents results to the user
*******************************************************************-->
<xsd:element name = "outputTask" substitutionGroup="anyTask">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="taskType">
        <xsd:sequence>
          <xsd:element ref = "var" minOccurs = "0" maxOccurs = "unbounded"/>
          <xsd:element ref = "input" minOccurs = "0" maxOccurs = "1"/>

          <xsd:choice>
            <xsd:element ref = "outputFile" minOccurs = "0" maxOccurs =
"unbounded"/>
            <xsd:element ref = "outputStatic" minOccurs = "0" maxOccurs =
"unbounded"/>
            <xsd:element ref = "xslt" minOccurs = "0" maxOccurs = "unbounded"/>
          </xsd:choice>

        </xsd:sequence>

      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<!--*******************************************************************
     Individual Task - Wrapper Task
        - accesses an external resource
```

```
***************************************************************-->
<xsd:element name = "wrapperTask" substitutionGroup="anyTask">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="taskType">
        <xsd:sequence>
          <xsd:element ref = "var" minOccurs = "0" maxOccurs = "unbounded"/>
          <xsd:element ref = "input" minOccurs = "1" maxOccurs = "1"/>
          <xsd:element ref = "output" minOccurs = "1" maxOccurs = "1"/>
          <xsd:element ref = "wrapper" minOccurs = "1" maxOccurs = "1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<!--****************************************************************
     Individual Task - Internal Task
        - performs non-interactive processing
***************************************************************-->
<xsd:element name = "internalTask" substitutionGroup="anyTask">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="taskType">
        <xsd:sequence>
          <xsd:element ref = "var" minOccurs = "0" maxOccurs = "unbounded"/>
          <xsd:element ref = "input" minOccurs = "0" maxOccurs = "1"/>
          <xsd:element ref = "output" minOccurs = "0" maxOccurs = "1"/>
          <xsd:element ref = "xslt" minOccurs = "0" maxOccurs = "unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<!--****************************************************************
     Individual Task - Profile Task
        - performs profile application
***************************************************************-->
<xsd:element name = "profileTask" substitutionGroup="anyTask">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="taskType">
        <xsd:sequence>
          <xsd:element ref = "input" minOccurs = "1" maxOccurs = "1"/>
          <xsd:element ref = "buildProfileFrom" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>


<!--****************************************************************
     High level task statement descriptions
        - subtask elements are contained by grouping tasks
***************************************************************-->
<xsd:element name = "subtask">
  <xsd:complexType>
    <xsd:attribute name="taskref" type="nonEmptyString" use="required"/>
    <xsd:attribute name="name" type="nonEmptyString" use="required"/>
    <xsd:attribute name="sequence" type="xsd:integer" use="optional"
default="0"/>
    <xsd:attribute name="required" type="xsd:boolean" use="optional"
default="false"/>
    <xsd:attribute name="activation" type="activationType" use="optional"
default="non_auto"/>
  </xsd:complexType>
</xsd:element>

<!--****************************************************************
     Task Statement - Output Task Statement
          - holds info about variables that this task outputs
****************************************************************-->
<xsd:element name = "output">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref = "infoOut" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!--****************************************************************
     Task Statement - Input Task Statement
          - holds info about variables that are input to this task
****************************************************************-->
<xsd:element name = "input">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref = "infoIn" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!--****************************************************************
     Task Statement - Xslt Task Statement
          - applies specified xslt(stylesheet) to specified xml
          - contains in - where to get the  input from
                     xsl - which spreadsheet to apply to that input
                     out - where to put the output

Note there can be multiple input sources, each one is converted
to an element, the first one is used as the source document
for the xsl application and all the rest are passed as parameters.

The first one does not receive the following treatement it is just
used as is like before.

The parameters are as follows the name of the variable will be used
as the name of the parameter if the data is loaded from a variable.
If the data is loaded from a file path contained in a variable
the parameter will also have the same name as the variable.
If the data is loaded from a file path specified as the contents
of the in element then the parameter will have the name param1 for the
first param2 for the second and so on. Numbers will be skipped
if any of the defined variables have the name of the same form,
therefore if a variable called param1 exists then the first
in statement holding an immediate file path will be called param2
not param1.
****************************************************************-->
<xsd:element name = "xslt">
  <xsd:complexType>
```

```
    <xsd:complexContent>
      <xsd:extension base="remoteExecutable">
        <xsd:sequence>
          <xsd:element ref = "in" minOccurs = "0" maxOccurs = "unbounded"/>
          <xsd:element ref = "xsl" minOccurs = "1" maxOccurs = "1"/>
          <xsd:element ref = "out" minOccurs = "0" maxOccurs = "1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<!--****************************************************************
     Task Statement - Autogen Task Statement
        - automatically creates an input stylesheet from the values
        - then applies the stylesheet to specified xml
****************************************************************-->
<xsd:element name = "autogen">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="remoteExecutable">
        <xsd:sequence>
          <xsd:element ref="header" minOccurs="0" maxOccurs="1"/>
          <xsd:element ref="footer" minOccurs="0" maxOccurs="1"/>
          <xsd:element ref="submit" minOccurs="0" maxOccurs="1"/>
          <xsd:element ref="autoVar" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "header" type="nonEmptyString"/>
<xsd:element name = "footer" type="nonEmptyString"/>
<xsd:element name = "submit" type="nonEmptyString"/>

<xsd:element name = "autoVar">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="nameDisplay" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="domainType" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="varSequence" minOccurs="0" maxOccurs="1"/>
      <xsd:choice>
        <xsd:element ref="textDisplay" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="dropDisplay" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="radioDisplay" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="checkDisplay" minOccurs="0" maxOccurs="1"/>
      </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="name" type="nonEmptyString" use="required"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "nameDisplay" type="nonEmptyString"/>
<xsd:element name = "domainType" type="nonEmptyString"/>
<xsd:element name = "varSequence" type="xsd:integer"/>

<xsd:element name = "textDisplay">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="size" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "size" type="xsd:integer"/>

<xsd:element name = "dropDisplay">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="entry" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "radioDisplay">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="entry" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "entry">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="entryValue" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="entryWording" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "entryValue" type="nonEmptyString"/>
<xsd:element name = "entryWording" type="nonEmptyString"/>

<xsd:element name = "checkDisplay">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="entryWording" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>


<!--***************************************************************
     Task Statement - OutputFile Task Statement
         - outputs the contents of the specified file
***************************************************************-->
<xsd:element name = "outputFile">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="remoteExecutable">
        <xsd:sequence>
          <xsd:element ref = "filePath" minOccurs = "1" maxOccurs =
"unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "outputStatic">
```

```
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="remoteExecutable">
        <xsd:sequence>
          <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "var">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="remoteExecutable">
        <xsd:sequence>
          <xsd:element ref = "varValue" minOccurs = "0" maxOccurs = "1"/>
        </xsd:sequence>
        <xsd:attribute name = "name" type="nonEmptyString" use="required"/>
        <xsd:attribute name = "type" type="nonEmptyString" use="required"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "wrapper">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="remoteExecutable">
        <xsd:sequence>
          <xsd:element ref = "wrapperName" minOccurs = "1" maxOccurs = "1"/>
          <xsd:element ref = "wrapperInput" minOccurs = "1" maxOccurs = "1"/>
          <xsd:element ref = "wrapperOutput" minOccurs = "1" maxOccurs = "1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>


<xsd:element name = "domain" type="nonEmptyString"/>

<!--****************************************************************
 Conceptual Constraints - filepath to a stylesheet with the domain constraints
****************************************************************-->
<xsd:element name = "conceptualConstraints" type="nonEmptyString"/>

<xsd:element name = "modelDescription" type="xsd:string"/>


<!--****************************************************************
 Functional Constraints - filepath to a stylesheet with the task constraints
****************************************************************-->
<xsd:element name = "functionalConstraints" type="nonEmptyString"/>

<xsd:element name = "name" type="nonEmptyString"/>
<!--
<xsd:element name = "description" type="xsd:string"/>
-->
<xsd:element name = "description">
```

```
  <xsd:complexType>
    <xsd:sequence>
      <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "agent" type="nonEmptyString"/>

<xsd:element name = "infoName" type="nonEmptyString"/>

<xsd:element name = "xpath" type="nonEmptyString"/>

<!--*****************************************************************
     Information elements - store info about which types of
         data are to be stored where
*****************************************************************-->
<xsd:element name = "infoOut">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="info">
        <xsd:attribute name="writemode" type="writemodeType" use="optional"
default="overwrite"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "infoIn">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="info">
        <xsd:attribute name="type" type="nonEmptyString" use="required"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<!--*****************************************************************
     Task Statement Component - in
        - where to get the input from
        - there are 3 options:
        1. if filePath is true - value is a filePath to a file
        2. if var is true - value is the var(variable) from
                  which to load the data
        3. if BOTH filePath and var are true - the value of the
              specified var is taken to be a filePath
              (the variable must be of type filePath)
*****************************************************************-->
<xsd:element name = "in">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base = "nonEmptyString">
        <xsd:attribute name="filePath" type="xsd:boolean" use="optional"
default="false"/>
        <xsd:attribute name="var" type="xsd:boolean" use="optional"
default="false"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

```
<!--****************************************************************
     Task Statement Component - xsl
        - which spreadsheet to apply to that input
        - there are 3 options:
        1. if filePath is true - value is a filePath to a file
        2. if var is true - value is the var(variable) from
                           which to load the data
        3. if BOTH filePath and var are true - the value of the
                specified var is taken to be a filePath
                (the variable must be of type filePath)
*************************************************************-->
<xsd:element name = "xsl">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base = "nonEmptyString">
        <xsd:attribute name="filePath" type="xsd:boolean" use="optional"
default="false"/>
        <xsd:attribute name="var" type="xsd:boolean" use="optional"
default="false"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>

<!--****************************************************************
     Task Statement Component - out
        - where to put the output
        - this may be omitted, in which case the application
          output goes to the screen (user display)
        - there are 4 options if not omitted:
        1. if filePath is true - value is a filePath to a file
        2. if var is true - value is the var(variable) to
                           which data is stored
        3. if BOTH filePath and var are true - the value of
                  the specified var is taken to be a filePath
                  (the variable must be of type filePath)
        4. if BOTH filePath and var are false(same as omitted)
             - application output goes to the screen
*************************************************************-->
<xsd:element name = "out">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base = "nonEmptyString">
        <xsd:attribute name="filePath" type="xsd:boolean" use="optional"
default="false"/>
        <xsd:attribute name="var" type="xsd:boolean" use="optional"
default="false"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name = "filePath" type="nonEmptyString"/>

<xsd:element name = "varValue">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:any processContents="skip" />
    </xsd:sequence>
  </xsd:complexType>
```

```
</xsd:element>

<xsd:element name = "wrapperName" type="nonEmptyString"/>

<xsd:element name = "wrapperInput" type="nonEmptyString"/>

<xsd:element name = "wrapperOutput" type="nonEmptyString"/>

<xsd:element name = "link" type="nonEmptyString"/>

<xsd:element name = "clearActivation" type="nonEmptyString"/>

<xsd:element name = "buildProfileFrom" type="nonEmptyString"/>


<!--***************************************************************
     Task Component - taskType complexType
         - used to derive tasks by extension
         - contains the elements common to all tasks
         - tasks currently derived are groupingTask, inputTask
           outputTask, wrapperTask, and internalTask
*************************************************************-->
<xsd:complexType name = "taskType">
  <xsd:sequence>
    <xsd:element ref = "name" minOccurs = "1" maxOccurs = "1"/>
    <xsd:element ref = "description" minOccurs = "1" maxOccurs = "1"/>
    <xsd:element ref = "conceptualConstraints" minOccurs = "0" maxOccurs =
"1"/>
    <xsd:element ref = "functionalConstraints" minOccurs = "0" maxOccurs =
"1"/>
    <xsd:element ref = "agent" minOccurs = "0" maxOccurs = "unbounded"/>
    <xsd:element ref = "link" minOccurs = "0" maxOccurs = "unbounded"/>
    <xsd:element ref = "clearActivation" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="taskid" type="nonEmptyString" use="required"/>
</xsd:complexType>

<!--***************************************************************
     Task Statement Component - remoteExecutable complexType
         - used to derive remote executable task statements by extension
         - contains the elements common to all remote executable tasks
statements

If the executeRemotely flag is set then the statement is forced to throw
a DefferedExecutionException regardless of the statements abilitity to be
completed locally.

Please note that the external id must be a three part string, containing
task model name|task id|external id of task statement, with the '|' being
the delimiters.
*************************************************************-->
<xsd:complexType name = "remoteExecutable">
  <xsd:sequence />
  <xsd:attribute name="externalId" type="nonEmptyString" use="optional"
default=""/>
  <xsd:attribute name="executeRemotely" type="xsd:boolean" use="optional"
default="false"/>
</xsd:complexType>

<!--***************************************************************
     Information Component - info complexType
```

```
        - used to derive information elements by extension
        - contains the elements common to all info elements
        - info elements currently derived are infoIn and infoOut
****************************************************************-->
<xsd:complexType name = "info">
  <xsd:sequence>
    <xsd:element ref = "infoName" minOccurs = "1" maxOccurs = "1"/>
    <xsd:element ref = "xpath" minOccurs = "1" maxOccurs = "1"/>
  </xsd:sequence>
</xsd:complexType>

<!--****************************************************************
     Task Statement Component - writemodeType
        - this type describes the different writemodes available
****************************************************************-->
<xsd:simpleType name = "writemodeType">
  <xsd:restriction base = "xsd:string">
    <xsd:enumeration value = "append"/>
    <xsd:enumeration value = "overwrite"/>
    <xsd:enumeration value = "mergeRoot"/>
  </xsd:restriction>
</xsd:simpleType>

<!--****************************************************************
     Task Statement Component - activationType
        - describes the types of activation that a task may have
        - automatic or non-automatic
****************************************************************-->
<xsd:simpleType name = "activationType">
  <xsd:restriction base = "xsd:string">
    <xsd:enumeration value = "auto"/>
    <xsd:enumeration value = "non_auto"/>
    <xsd:enumeration value = "only_manual"/>
  </xsd:restriction>
</xsd:simpleType>

<!--****************************************************************
     Basic Type - nonEmptyString
        - makes sure that if an element is specified then it at
          least has a value, if it does not the java code needs to be
          more complex as empty elements don't have child text nodes and
          this needs to be accounted for.
****************************************************************-->
<xsd:simpleType name="nonEmptyString">
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="1"/>
  </xsd:restriction>
</xsd:simpleType>


</xsd:schema>
```

2. Task Model Instance for Book Finding - XML Schema Instance

(from tamex/taskModels/myBookBuying/BookBuying.xml 04/07/03)

```xml
<?xml version="1.0"?>
<taskModel name="myBookBuying" start="T0"
      xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation='../../../config/TaskModel.xsd'>

<domain>../../../domainModels/BookBuying/BookBuyingDomain.xsd</domain>
<modelDescription>This task model allows you to view and manipulate books from
a variety of resources.</modelDescription>

<groupingTask taskid="T0">
<name>Root task</name>
<description>Select a task to perform</description>
<subtask taskref="T00" name="Select Profile Options" sequence="1"
activation="only_manual" />
<subtask taskref="T01" name="Specify Book Selection Criteria" sequence="1"
required="true" />
<subtask taskref="T02" name="Access Book-Selling Resources" sequence="2"
required="true" activation="auto"/>
<subtask taskref="T03" name="Sort Books" sequence="3"/>
<subtask taskref="T03b" name="Clear Sort Status" sequence="3"/>
<subtask taskref="T04" name="Group Books" sequence="3"/>
<subtask taskref="T05" name="Filter Books" sequence="3"/>
<subtask taskref="T06" name="View Current" sequence="3"
activation="only_manual" required="true" />
<subtask taskref="T07" name="Clear Results" sequence="3"
activation="only_manual" required="true" />
<subtask taskref="T08" name="Reload Original" sequence="3"
activation="only_manual" required="true" />
<subtask taskref="T09" name="Show Explorer Applet" sequence="3"
activation="only_manual" required="true" />
</groupingTask>

<groupingTask taskid="T00" min="0">
  <name>Select Profile Options</name>
  <description>Allows you to select a profile for use with this task
model</description>
  <subtask taskref="T001" name="Adapt Task Model" activation="only_manual"/>
</groupingTask>

<inputTask taskid="T001">
  <name>Adapt Task Model</name>
  <description>Allows you to modify the structure of the task model and enter
default info to be used in later steps</description>
  <!--
<functionalConstraints>../../../taskModels/myBookBuying/profileOptionsConstrain
ts.xsl</functionalConstraints> file not yet exists on disk -->
  <link>profile_applier</link>
  <var name="profileOptions" type="profileOptionsContainer"/>
  <var name="defaults" type="defaultsContainer"/>
  <output>
    <infoOut writemode="overwrite">
      <infoName>profileOptions</infoName>
      <xpath>/profile</xpath>
    </infoOut>
    <infoOut writemode="overwrite">
      <infoName>defaults</infoName>
      <xpath>/default</xpath>
    </infoOut>
  </output>
  <xslt>
    <in filePath="true">../../../taskModels/myBookBuying/BookBuying.xml</in>
    <xsl filePath="true">../../../config/genProfileInput.xsl</xsl>
  </xslt>
</inputTask>

<profileTask taskid="profile_applier">
  <name>Profile application task</name>
  <description>This Task applys profiles, as in it implements the profile
mechanisim</description>
  <input>
    <infoIn type="profileOptionsContainer">
      <infoName>profileOptions</infoName>
      <xpath>/profile</xpath>
    </infoIn>
  </input>
  <buildProfileFrom>profileOptions</buildProfileFrom>
</profileTask>

<inputTask taskid="T01">
  <name>Specify Book Selection Criteria</name>
  <description>Enter your input</description>
  <link>ClearResourceAccess</link>
  <var name="t01query" type="BookQuery"/>
  <output>
    <infoOut writemode="overwrite">
      <infoName>t01query</infoName>
      <xpath>/BookQuery1</xpath>
    </infoOut>
  </output>

<!--
  <xslt externalId="myBookBuying|T01|R01">
    <xsl filePath="true">../../../taskModels/myBookBuying/t01input.xsl</xsl>
  </xslt>
-->

<autogen>
    <header>Please enter the keyword that you would like the book buying
resources to look for.</header>
    <submit>Submit your book request</submit>
    <autoVar name="t01query">
      <domainType>/BookQuery</domainType>
      <textDisplay>
        <size>24</size>
      </textDisplay>
    </autoVar>
  </autogen>

</inputTask>

<groupingTask taskid="T02" min="1">
<name>Access Book-Selling Resources</name>
<description>Select the resources wanted</description>
<subtask taskref="T021" name="Access Amazon" />
```

```xml
<subtask taskref="T022" name="Access Chapters" />
<subtask taskref="T023" name="Access 1Book" />
</groupingTask>

<wrapperTask taskid="T021">
  <name>Access Amazon</name>
  <description>Access the Amazon website</description>

<conceptualConstraints>../../../taskModels/myBookBuying/priceCconstraints.xsl</
conceptualConstraints>
  <agent>eddie01</agent>
  <link>FixData</link>
  <link>ClearSorting</link>
  <link>ClearGrouping</link>
  <link>ClearFiltering</link>
  <var name="result" type="searchResult"/>
  <input>
    <infoIn type="BookQuery">
      <infoName>aQuery</infoName>
      <xpath>/BookQuery1</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="mergeRoot">
      <infoName>result</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <wrapper externalId="myBookBuying|T021|S01">
    <wrapperName>book-amazon</wrapperName>
    <wrapperInput>aQuery</wrapperInput>
    <wrapperOutput>result</wrapperOutput>
  </wrapper>
</wrapperTask>

<wrapperTask taskid="T022">
  <name>Access Chapters</name>
  <description>Access the Chapters website</description>

<conceptualConstraints>../../../taskModels/myBookBuying/priceCconstraints.xsl</
conceptualConstraints>
  <agent>A01</agent>
  <agent>A02</agent>
  <link>FixData</link>
  <link>ClearSorting</link>
  <link>ClearGrouping</link>
  <link>ClearFiltering</link>
  <var name="result" type="searchResult"/>
  <input>
    <infoIn type="BookQuery">
      <infoName>aQuery</infoName>
      <xpath>/BookQuery1</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="mergeRoot">
      <infoName>result</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <wrapper>
```

```
    <wrapperName>book-chapters</wrapperName>
    <wrapperInput>aQuery</wrapperInput>
    <wrapperOutput>result</wrapperOutput>
  </wrapper>
</wrapperTask>

<wrapperTask taskid="T023">
  <name>Access 1Book</name>
  <description>Access the 1Book website</description>

<conceptualConstraints>../../../taskModels/myBookBuying/priceCconstraints.xsl</
conceptualConstraints>
  <agent>A01</agent>
  <agent>A02</agent>
  <link>FixData</link>
  <link>ClearSorting</link>
  <link>ClearGrouping</link>
  <link>ClearFiltering</link>
  <var name="result" type="searchResult"/>
  <input>
    <infoIn type="BookQuery">
      <infoName>aQuery</infoName>
      <xpath>/BookQuery1</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="mergeRoot">
      <infoName>result</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <wrapper>
    <wrapperName>book-1book</wrapperName>
    <wrapperInput>aQuery</wrapperInput>
    <wrapperOutput>result</wrapperOutput>
  </wrapper>
</wrapperTask>

<groupingTask taskid="T03" max="1" min="1">
<name>Sort Books</name>
<description>Select another task</description>
<link>T06</link>
<subtask taskref="T031" name="Sort by Title" />
<subtask taskref="T032" name="Sort by Author" />
<subtask taskref="T033" name="Sort by Type" />
<subtask taskref="T034" name="Sort by Price" />
<subtask taskref="T035" name="Sort by Publisher" />
<subtask taskref="T036" name="Sort by Origin" />
</groupingTask>

<internalTask taskid="T031">
  <name>Sort by Title</name>
  <description>Sorts book info by title</description>
  <link>T06</link>
  <clearActivation>T032</clearActivation>
  <clearActivation>T033</clearActivation>
  <clearActivation>T034</clearActivation>
  <clearActivation>T035</clearActivation>
  <clearActivation>T036</clearActivation>
  <var name="criteria" type="sortCriteria">
    <varValue><sortCriteria>title</sortCriteria></varValue>
  </var>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">criteria</in>
    <xsl filePath="true">../../../taskModels/myBookBuying/sortGeneral.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<internalTask taskid="T032">
  <name>Sort by Author</name>
  <description>Sorts book info by Author</description>
  <link>T06</link>
  <clearActivation>T031</clearActivation>
  <clearActivation>T033</clearActivation>
  <clearActivation>T034</clearActivation>
  <clearActivation>T035</clearActivation>
  <clearActivation>T036</clearActivation>
  <var name="criteria" type="sortCriteria">
    <varValue><sortCriteria>author</sortCriteria></varValue>
  </var>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">criteria</in>
    <xsl filePath="true">../../../taskModels/myBookBuying/sortGeneral.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<internalTask taskid="T033">
  <name>Sort by Type</name>
  <description>Sorts book info by Type</description>
  <link>T06</link>
  <clearActivation>T031</clearActivation>
  <clearActivation>T032</clearActivation>
  <clearActivation>T034</clearActivation>
  <clearActivation>T035</clearActivation>
  <clearActivation>T036</clearActivation>
  <var name="criteria" type="sortCriteria">
    <varValue><sortCriteria>type</sortCriteria></varValue>
  </var>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">criteria</in>
    <xsl filePath="true">../../../taskModels/myBookBuying/sortGeneral.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<internalTask taskid="T03b">
  <name>Clear Sort Status</name>
  <description>Clears the sorted status of your current result
set.</description>
  <link>T06</link>
  <clearActivation>T031</clearActivation>
  <clearActivation>T032</clearActivation>
  <clearActivation>T033</clearActivation>
  <clearActivation>T034</clearActivation>
  <clearActivation>T035</clearActivation>
  <clearActivation>T036</clearActivation>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/clearSortStatus.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<internalTask taskid="T034">
  <name>Sort by Price</name>
  <description>Sorts book info by Price</description>
  <link>T06</link>
  <clearActivation>T031</clearActivation>
  <clearActivation>T032</clearActivation>
  <clearActivation>T033</clearActivation>
  <clearActivation>T035</clearActivation>
  <clearActivation>T036</clearActivation>
  <var name="criteria" type="sortCriteria">
    <varValue><sortCriteria>price</sortCriteria></varValue>
  </var>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">criteria</in>
    <xsl filePath="true">../../../taskModels/myBookBuying/sortGeneral.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<internalTask taskid="T035">
  <name>Sort by Publisher</name>
  <description>Sorts book info by Publisher</description>
  <link>T06</link>
  <clearActivation>T031</clearActivation>
  <clearActivation>T032</clearActivation>
  <clearActivation>T033</clearActivation>
  <clearActivation>T034</clearActivation>
  <clearActivation>T036</clearActivation>
  <var name="criteria" type="sortCriteria">
    <varValue><sortCriteria>publisher</sortCriteria></varValue>
  </var>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">criteria</in>
    <xsl filePath="true">../../../taskModels/myBookBuying/sortGeneral.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<internalTask taskid="T036">
  <name>Sort by Origin</name>
  <description>Sorts book info by Origin</description>
  <link>T06</link>
  <clearActivation>T031</clearActivation>
  <clearActivation>T032</clearActivation>
  <clearActivation>T033</clearActivation>
  <clearActivation>T034</clearActivation>
  <clearActivation>T035</clearActivation>
  <var name="criteria" type="sortCriteria">
    <varValue><sortCriteria>origin</sortCriteria></varValue>
  </var>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">criteria</in>
    <xsl filePath="true">../../../taskModels/myBookBuying/sortGeneral.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<groupingTask taskid="T04">
  <name>Group Books</name>
  <description>Routines that group books according to various
categories</description>
  <link>T06</link>
  <subtask taskref="T041" name="Group by Type"/>
  <subtask taskref="T042" name="Group by Author"/>
  <subtask taskref="T043" name="Group by Origin"/>
  <subtask taskref="T044" name="Group by Publisher"/>
</groupingTask>

<internalTask taskid="T041">
  <name>Group by Type</name>
  <description>Groups book info by Type</description>
  <agent>A01</agent>
  <link>ClearSorting</link>
  <link>T06</link>
  <var name="criteria" type="groupCriteria">
    <varValue><groupCriteria>type</groupCriteria></varValue>
  </var>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt externalId="myBookBuying|T041|S01">
    <in var="true">aSearchResult</in>
    <in var="true">criteria</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/groupGeneralFake.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<internalTask taskid="T042">
  <name>Group by Author</name>
  <description>Groups book info by Author</description>
  <link>ClearSorting</link>
  <link>T06</link>
  <var name="criteria" type="groupCriteria">
    <varValue><groupCriteria>author</groupCriteria></varValue>
  </var>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">criteria</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/groupGeneral.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<internalTask taskid="T043">
  <name>Group by Origin</name>
  <description>Groups book info by Site of origin.</description>
  <link>ClearSorting</link>
  <link>T06</link>
  <var name="criteria" type="groupCriteria">
    <varValue><groupCriteria>origin</groupCriteria></varValue>
  </var>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">criteria</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/groupGeneral.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<internalTask taskid="T044">
  <name>Group by Publisher</name>
  <description>Groups book info by Publisher</description>
  <link>ClearSorting</link>
  <link>T06</link>
  <var name="criteria" type="groupCriteria">
    <varValue><groupCriteria>publisher</groupCriteria></varValue>
  </var>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">criteria</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/groupGeneral.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<groupingTask taskid="T05">
  <name>Filter Books</name>
  <description>Filters out and in books.</description>
  <link>T06</link>
  <subtask taskref="T051" name="Filter by Type"/>
  <subtask taskref="T052" name="Filter by Origin"/>
  <subtask taskref="T053" name="Filter by Publisher"/>
  <subtask taskref="T054" name="Filter by Author"/>
  <subtask taskref="T055" name="Filter by Title"/>
  <subtask taskref="T056" name="Filter by Price"/>
</groupingTask>

<inputTask taskid="T051">
  <name>Filter by Type</name>
  <description>Enter the filter criteria.</description>
  <link>T051b</link>
  <var name="aTypeFilter" type="typeFilter"/>
  <var name="criteria" type="filterCriteria">
    <varValue><filterCriteria>type</filterCriteria></varValue>
  </var>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aTypeFilter</infoName>
      <xpath>/typeFilter</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">criteria</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/filterGeneral.xsl</xsl>
  </xslt>
</inputTask>

<internalTask taskid="T051b">
  <name>Filter by Type</name>
  <description>Filters book info by type.</description>
  <link>T06</link>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
    <infoIn type="typeFilter">
      <infoName>aFilter</infoName>
      <xpath>/typeFilter</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">aFilter</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/filterGeneralWork.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<!--
<outputTask taskid="filterByTypeMessage">
  <name>FilterByTypeMessage</name>
  <description>Displays a short message saying that the search results have
been filtered by type.</description>
  <outputStatic>and the Result set has been filtered by Type.</outputStatic>
</outputTask>
-->

<inputTask taskid="T052">
  <name>Filter by Origin</name>
  <description>Enter the filter criteria.</description>
  <link>T052b</link>
  <var name="anOriginFilter" type="originFilter"/>
  <var name="criteria" type="filterCriteria">
    <varValue><filterCriteria>origin</filterCriteria></varValue>
  </var>
  <output>
    <infoOut writemode="overwrite">
      <infoName>anOriginFilter</infoName>
      <xpath>/originFilter</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">criteria</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/filterGeneral.xsl</xsl>
  </xslt>
</inputTask>

<internalTask taskid="T052b">
  <name>Filter by Origin</name>
  <description>Filters book info by origin.</description>
  <link>T06</link>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
    <infoIn type="originFilter">
      <infoName>aFilter</infoName>
      <xpath>/originFilter</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">aFilter</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/filterGeneralWork.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<!--
<outputTask taskid="filterByOriginMessage">
  <name>Filter By Origin Message</name>
  <description>Displays a short message saying that the search results have
been filtered by Origin.</description>
  <outputStatic>and the Result set has been filtered by Origin.</outputStatic>
</outputTask>
-->

<inputTask taskid="T053">
  <name>Filter by Publisher</name>
  <description>Enter the filter criteria.</description>
  <link>T053b</link>
  <var name="aPublisherFilter" type="publisherFilter"/>
  <var name="criteria" type="filterCriteria">
    <varValue><filterCriteria>publisher</filterCriteria></varValue>
  </var>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aPublisherFilter</infoName>
      <xpath>/publisherFilter</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">criteria</in>
    <xsl
```

```
filePath="true">../../../taskModels/myBookBuying/filterGeneral.xsl</xsl>
  </xslt>
</inputTask>

<internalTask taskid="T053b">
  <name>Filter by Publisher</name>
  <description>Filters book info by Publisher.</description>
  <link>T06</link>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
    <infoIn type="publisherFilter">
      <infoName>aFilter</infoName>
      <xpath>/publisherFilter</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">aFilter</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/filterGeneralWork.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<!--
<outputTask taskid="filterByPublisherMessage">
  <name>FilterByPublisherMessage</name>
  <description>Displays a short message saying that the search results have
been filtered by publisher.</description>
  <outputStatic>and the Result set has been filtered by
Publisher.</outputStatic>
</outputTask>
-->

<inputTask taskid="T054">
  <name>Filter by Author</name>
  <description>Enter the filter criteria.</description>
  <link>T054b</link>
  <var name="anAuthorFilter" type="authorFilter"/>
  <var name="criteria" type="filterCriteria">
    <varValue><filterCriteria>author</filterCriteria></varValue>
  </var>
  <output>
    <infoOut writemode="overwrite">
      <infoName>anAuthorFilter</infoName>
      <xpath>/authorFilter</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">criteria</in>
    <xsl
```

```
filePath="true">../../../taskModels/myBookBuying/filterGeneral.xsl</xsl>
  </xslt>
</inputTask>

<internalTask taskid="T054b">
  <name>Filter by Author</name>
  <description>Filters book info by author.</description>
  <link>T06</link>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
    <infoIn type="authorFilter">
      <infoName>aFilter</infoName>
      <xpath>/authorFilter</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">aFilter</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/filterGeneralWork.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<!--
<outputTask taskid="filterByAuthorMessage">
  <name>FilterByAuthorMessage</name>
  <description>Displays a short message saying that the search results have
been filtered by author.</description>
  <outputStatic>and the Result set has been filtered by Author.</outputStatic>
</outputTask>
-->

<inputTask taskid="T055">
  <name>Filter by Title</name>
  <description>Enter the filter criteria.</description>
  <link>T055b</link>
  <var name="aTitleFilter" type="titleFilter"/>
  <var name="criteria" type="filterCriteria">
    <varValue><filterCriteria>title</filterCriteria></varValue>
  </var>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aTitleFilter</infoName>
      <xpath>/titleFilter</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">criteria</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/filterGeneral.xsl</xsl>
  </xslt>
```

```
</inputTask>

<internalTask taskid="T055b">
  <name>Filter by Title</name>
  <description>Filters book info by title.</description>
  <link>T06</link>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
    <infoIn type="titleFilter">
      <infoName>aFilter</infoName>
      <xpath>/titleFilter</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">aFilter</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/filterGeneralWork.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<!--
<outputTask taskid="filterByTitleMessage">
  <name>FilterByTitleMessage</name>
  <description>Displays a short message saying that the search results have
been filtered by title.</description>
  <outputStatic>and the Result set has been filtered by Title.</outputStatic>
</outputTask>
-->

<inputTask taskid="T056">
  <name>Filter by Price</name>
  <description>Enter the min and max of price range</description>

<conceptualConstraints>../../../taskModels/myBookBuying/filterByPriceCconstrain
ts.xsl</conceptualConstraints>

<functionalConstraints>../../../taskModels/myBookBuying/filterByPriceFconstrain
ts.xsl</functionalConstraints>
  <link>T056b</link>
  <var name="aPriceRange" type="priceRange"/>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aPriceRange</infoName>
      <xpath>/priceRange</xpath>
    </infoOut>
  </output>

<!--
  <xslt>
```

```
    <xsl
filePath="true">../../../taskModels/myBookBuying/filterByPrice.xsl</xsl>
  </xslt>
-->

  <autogen>
    <header>This form allows you to filter the results from the resources
according to their price.  Enter the min and max values for price range in the
form nnn.nn</header>
    <footer>Please note that this operation will match the price of the book
and will filter according to the price range entered.</footer>
    <submit>Submit filter criteria</submit>
    <autoVar name="aPriceRange">
      <nameDisplay>Minimum Book Price:</nameDisplay>
      <domainType>/priceRange/priceMin/price</domainType>
      <varSequence>1</varSequence>
      <textDisplay>
        <size>11</size>
      </textDisplay>
    </autoVar>
    <autoVar name="aPriceRange">
      <nameDisplay>Maximum Book Price:</nameDisplay>
      <domainType>/priceRange/priceMax/price</domainType>
      <varSequence>2</varSequence>
      <textDisplay>
        <size>11</size>
      </textDisplay>
    </autoVar>
  </autogen>

</inputTask>

<internalTask taskid="T056b">
  <name>Filter books by Price</name>
  <description>Filters book info by price.</description>

<functionalConstraints>../../../taskModels/myBookBuying/filterByPriceFconstrain
ts.xsl</functionalConstraints>
  <link>T06</link>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
    <infoIn type="priceRange">
      <infoName>aPriceRange</infoName>
      <xpath>/priceRange</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <in var="true">aPriceRange</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/filterByPriceWork.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<!--
<outputTask taskid="filterByPriceMessage">
  <name>FilterByPriceMessage</name>
  <description>Displays a short message saying that the search results have
been filtered by price.</description>
  <outputStatic>and the Result set has been filtered by Price.</outputStatic>
</outputTask>
-->

<outputTask taskid="T06">
  <name>View Current</name>
  <description>View current book list.</description>
  <input>
    <infoIn type="searchResult">
      <infoName>result</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <xslt>
    <in var="true">result</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/displayBooks.xsl</xsl>
  </xslt>
</outputTask>

<internalTask taskid="T07">
  <name>Clear Results</name>
  <description>Clears all of the current results.</description>
  <link>ClearResultsMessage</link>
  <link>ClearAlmostAllActivation</link>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/clearSearchResults.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<internalTask taskid="T08">
  <name>Reload Original</name>
  <description>Reload Original Book List</description>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <xsl
filePath="true">../../../taskModels/myBookBuying/reloadOriginalSet.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<outputTask taskid="ClearResultsMessage">
  <name>ClearResultsMessage</name>
  <description>Displays a short message saying that the search results have
been cleared.</description>
  <outputStatic>The book set has been cleared, you will need to access the
resources again in order to continue with this interaction.</outputStatic>
</outputTask>

<internalTask taskid="ClearAlmostAllActivation">
  <name>Clear Almost All Activations</name>
  <description>Clears the activation and started flags from all of the tasks
except for the input query one.</description>

  <link>ClearResourceAccess</link>
  <link>ClearSorting</link>
  <link>ClearGrouping</link>
  <link>ClearFiltering</link>

  <clearActivation>T06</clearActivation>
  <clearActivation>T08</clearActivation>
</internalTask>

<internalTask taskid="ClearAllActivation">
  <name>Clear All Activations</name>
  <description>Clears the activation and started flags from all of the
tasks.</description>

  <link>ClearAlmostAllActivation</link>
  <clearActivation>T01</clearActivation>
  <clearActivation>T07</clearActivation>
</internalTask>

<internalTask taskid="ClearResourceAccess">
  <name>Clear Resource Accesses</name>
  <description>Clears the activation of resource access tasks.</description>

  <clearActivation>T02</clearActivation>
  <clearActivation>T021</clearActivation>
  <clearActivation>T022</clearActivation>
  <clearActivation>T023</clearActivation>
</internalTask>

<internalTask taskid="ClearSorting">
  <name>Clear Sorting Tasks</name>
  <description>Clears the activation of sorting tasks.</description>
  <clearActivation>T03</clearActivation>
  <clearActivation>T031</clearActivation>
  <clearActivation>T032</clearActivation>
  <clearActivation>T033</clearActivation>
  <clearActivation>T034</clearActivation>
  <clearActivation>T035</clearActivation>
  <clearActivation>T036</clearActivation>
</internalTask>

<internalTask taskid="ClearGrouping">
  <name>Clear Grouping Tasks</name>
  <description>Clears the activation of grouping tasks.</description>

  <clearActivation>T04</clearActivation>
  <clearActivation>T041</clearActivation>
  <clearActivation>T042</clearActivation>
  <clearActivation>T043</clearActivation>
  <clearActivation>T044</clearActivation>
</internalTask>

<internalTask taskid="ClearFiltering">
  <name>Clear Filtering Tasks</name>
  <description>Clears the activation of filtering tasks.</description>

  <clearActivation>T05</clearActivation>
  <clearActivation>T051</clearActivation>
  <clearActivation>T052</clearActivation>
  <clearActivation>T053</clearActivation>
</internalTask>

<internalTask taskid="FixData">
  <name>Fix data</name>
  <description>Makes sure that wrapper data is of a valid format and
coolated.</description>
  <input>
    <infoIn type="searchResult">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <output>
    <infoOut writemode="overwrite">
      <infoName>aSearchResult</infoName>
      <xpath>/results</xpath>
    </infoOut>
  </output>
  <xslt>
    <in var="true">aSearchResult</in>
    <xsl filePath="true">../../../taskModels/myBookBuying/fixData.xsl</xsl>
    <out var="true">aSearchResult</out>
  </xslt>
</internalTask>

<outputTask taskid="T09">
  <name>Show Explorer Applet</name>
  <description>Shows an applet that allows you to view and categorize the
books.</description>
  <input>
    <infoIn type="searchResult">
      <infoName>result</infoName>
      <xpath>/results</xpath>
    </infoIn>
  </input>
  <xslt>
    <in var="true">result</in>
    <xsl filePath="true">../../../taskModels/myBookBuying/showApplet.xsl</xsl>
  </xslt>
</outputTask>

</taskModel>
```

3. Filter-by-Price Task Functional Constraint-checking Stylesheet

(from tamex/taskModels/myBookBuying/filterByPriceFconstraints.xsl   01/23/03)

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml"/>
<xsl:template match="/">
   <xsl:if test="number(/Vars/aPriceRange/priceRange/priceMin/price) &gt;
number(/Vars/aPriceRange/priceRange/priceMax/price)">
      <xsl:text> The minimum price is greater than the maximum price
</xsl:text>
      <xsl:text> The minimum price is  </xsl:text>
      <xsl:value-of select="/Vars/aPriceRange/priceRange/priceMin/price"/>
      <xsl:text> The maximum price is  </xsl:text>
      <xsl:value-of select="/Vars/aPriceRange/priceRange/priceMax/price"/>
   </xsl:if>

</xsl:template>
</xsl:stylesheet>
```

C - Profile Model

1. Profile Adaptation Stylesheet for Generating the Input Form

(from tamex/config/genProfileInput.xsl 01/23/03)

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- This stylesheet generates an input form for profile options given
     a task model. -->
<xsl:variable name="root" select="/"/>


<xsl:template match="/">

  <center><H3>Profile Options</H3></center>
  <br/>
  <p>
  Please select from the following options in order to build up your profile.
  </p>

  <!-- start the table that will structure the input form -->
  <table width="100%" border="true">

  <!-- add the intro -->
  <tr>
    <td width="5%"><center><b>Remove</b></center></td>
    <td width="20%"><center><b>Task Name</b></center></td>
    <td width="5%"><center><b>Automate</b></center></td>
    <td width="20%"><center><b>Messages / Enter Default
Information</b></center></td>
  </tr>

  <!-- find the start task and apply task template to it -->
  <xsl:variable name="theTask"
select="/taskModel/*[@taskid=/taskModel/@start]"/>
  <xsl:call-template name="processTask">
    <xsl:with-param name="theTask" select="$theTask"/>
  </xsl:call-template>

  </table> <!-- end of the top most table -->

  <!-- Add the submit and clear buttons -->
  <center>
    <input TYPE="submit" VALUE="Submit profile options"></input>
    <input TYPE="reset" VALUE="Clear"></input><p></p>
  </center>
</xsl:template>

<!-- This template processes a task, by examining subtasks if this is a
     grouping task and creating table rows -->
<xsl:template name="processTask">
  <xsl:param name="theTask"/>

  <!-- for each subtask task statement find the task and process it -->
  <xsl:for-each select="$theTask/subtask">
```

```
    <xsl:variable name="taskRef" select="./@taskref"/>
    <!-- get the task that this subtask task statement points to -->
    <xsl:variable name="aTask" select="$root/taskModel/*[@taskid=$taskRef]"/>


    <!-- start a new row for the task stmt currently selected. -->
    <xsl:call-template name="createRow">
       <xsl:with-param name="theTaskStmt" select="./"/>
    </xsl:call-template>


    <!-- recurse depth first -->
    <xsl:call-template name="processTask">
      <xsl:with-param name="theTask" select="$aTask"/>
    </xsl:call-template>
  </xsl:for-each> <!-- for each subtask end -->


</xsl:template>


<!-- This template creates a row in the table for the given task -->
<xsl:template name="createRow">
  <xsl:param name="theTaskStmt"/>

    <!-- if this task stmt points to a grouping task put in a
         blank row to seperate it from the others. -->

    <!-- get the task itself -->
    <xsl:variable name="taskRef" select="$theTaskStmt/@taskref"/>
    <xsl:variable name="theTask" select="$root/taskModel/*[@taskid =
$taskRef]"/>

    <xsl:if test="name($theTask) = 'groupingTask'">
    <tr>
     <td><hr width="100%"/></td>
     <td><hr width="100%"/></td>
     <td><hr width="100%"/></td>
     <td><hr width="100%"/></td>
    </tr>
    </xsl:if> <!-- end condition if this is a grouping task -->


    <tr> <!-- one row for each task stmt -->

      <!-- add the remove cell -->
      <xsl:call-template name="createRemoveCell">
        <xsl:with-param name="theTaskStmt" select="$theTaskStmt"/>
      </xsl:call-template>

      <!-- add the task name cell -->
      <xsl:call-template name="createNameCell">
        <xsl:with-param name="theTaskStmt" select="$theTaskStmt"/>
      </xsl:call-template>

      <!-- add the automate cell -->
      <xsl:call-template name="createAutomateCell">
        <xsl:with-param name="theTaskStmt" select="$theTaskStmt"/>
      </xsl:call-template>

      <!-- add the automate message -->
      <xsl:call-template name="createMessageDefaultCell">
        <xsl:with-param name="theTaskStmt" select="$theTaskStmt"/>
      </xsl:call-template>

    </tr>

</xsl:template>



<!-- will add the contents of the remove cell of the row for the given task -->
<!-- if this task may not be removed then the cell will be empty -->
<xsl:template name="createRemoveCell">
  <xsl:param name="theTaskStmt"/> <!-- the subtask task statement -->

  <td width="5%"> <!-- start of cell --><xsl:text>   </xsl:text>

    <!-- if required is not true then output the remove cell other wise -->
    <!-- leave it empty -->
    <xsl:if test="not($theTaskStmt/@required = 'true')">

      <!-- The checkbox -->
      <xsl:element name="input">
        <xsl:attribute
name="type"><xsl:text>checkbox</xsl:text></xsl:attribute>
        <xsl:attribute name="name">
          <xsl:text>profileOptions /profileOptionsContainer/rem</xsl:text>
          <xsl:value-of select="$theTaskStmt/@taskref"/>
          <xsl:text>/xsl</xsl:text>
        </xsl:attribute>
        <xsl:attribute name="value">
          <xsl:text>../../../config/removeTask.xsl</xsl:text>
        </xsl:attribute>
      </xsl:element>


      <!-- The name of the option (remove in this case) -->
      <xsl:element name="input">
        <xsl:attribute name="type"><xsl:text>hidden</xsl:text></xsl:attribute>
        <xsl:attribute name="name">
          <xsl:text>profileOptions /profileOptionsContainer/rem</xsl:text>
          <xsl:value-of select="$theTaskStmt/@taskref"/>
          <xsl:text>/name 1</xsl:text>
        </xsl:attribute>
        <xsl:attribute name="value">
          <xsl:text>remove</xsl:text>
        </xsl:attribute>
      </xsl:element>


      <!-- The value of the option (the task id of the task to be removed -->
      <xsl:element name="input">
        <xsl:attribute name="type"><xsl:text>hidden</xsl:text></xsl:attribute>
        <xsl:attribute name="name">
          <xsl:text>profileOptions /profileOptionsContainer/rem</xsl:text>
          <xsl:value-of select="$theTaskStmt/@taskref"/>
          <xsl:text>/value 2</xsl:text>
        </xsl:attribute>
        <xsl:attribute name="value">
          <xsl:value-of select="$theTaskStmt/@taskref"/>
        </xsl:attribute>
      </xsl:element>


    </xsl:if>

  </td> <!-- end of cell -->

</xsl:template>


<!-- will add the task name cell of the row for the given task -->
<xsl:template name="createNameCell">
  <xsl:param name="theTaskStmt"/> <!-- the subtask task statement -->

  <td width="20%"> <!-- start of cell --><xsl:text>   </xsl:text>
    <xsl:value-of select="$theTaskStmt/@name"/>
    <xsl:text> (</xsl:text>
    <xsl:value-of select="$theTaskStmt/@taskref"/>
    <xsl:text>)</xsl:text>
  </td> <!-- end of cell -->

</xsl:template>

<!-- will add the contents of the automate cell of the row for the given task -
->
<!-- if this task may not be automated then the cell will be empty -->
<xsl:template name="createAutomateCell">
  <xsl:param name="theTaskStmt"/> <!-- the subtask task statement -->

  <td width="5%"> <!-- start of cell --><xsl:text>   </xsl:text>
    <xsl:if test="not($theTaskStmt/@activation = 'only_manual') and
not($theTaskStmt/@activation = ‘auto’)">

      <!-- The checkbox -->
      <xsl:element name="input">
        <xsl:attribute
name="type"><xsl:text>checkbox</xsl:text></xsl:attribute>
        <xsl:attribute name="name">
          <xsl:text>profileOptions /profileOptionsContainer/auto</xsl:text>
          <xsl:value-of select="$theTaskStmt/@taskref"/>
          <xsl:text>/xsl</xsl:text>
        </xsl:attribute>
        <xsl:attribute name="value">
          <xsl:text>../../../config/automateTask.xsl</xsl:text>
        </xsl:attribute>
      </xsl:element>


      <!-- The name of the option (automate in this case) -->
      <xsl:element name="input">
        <xsl:attribute name="type"><xsl:text>hidden</xsl:text></xsl:attribute>
        <xsl:attribute name="name">
          <xsl:text>profileOptions /profileOptionsContainer/auto</xsl:text>
          <xsl:value-of select="$theTaskStmt/@taskref"/>
          <xsl:text>/name 1</xsl:text>
        </xsl:attribute>
        <xsl:attribute name="value">
          <xsl:text>automate</xsl:text>
        </xsl:attribute>
      </xsl:element>
      <!-- The value of the option (the task id of the task to be automated) --
>
      <xsl:element name="input">
        <xsl:attribute name="type"><xsl:text>hidden</xsl:text></xsl:attribute>
        <xsl:attribute name="name">
          <xsl:text>profileOptions /profileOptionsContainer/auto</xsl:text>
          <xsl:value-of select="$theTaskStmt/@taskref"/>
          <xsl:text>/value 2</xsl:text>
        </xsl:attribute>
        <xsl:attribute name="value">
          <xsl:value-of select="$theTaskStmt/@taskref"/>
        </xsl:attribute>
      </xsl:element>


    </xsl:if>
  </td> <!-- end of cell -->

</xsl:template>

<!-- will add the contents of the automate message cell of the row for the
given task -->
<!-- this message will only be added if it makes sense for this particular
group -->
<xsl:template name="createMessageDefaultCell">
  <xsl:param name="theTaskStmt"/> <!-- the subtask task statement -->

  <!-- get the task itself -->
  <xsl:variable name="taskRef" select="$theTaskStmt/@taskref"/>
  <xsl:variable name="theTask" select="$root/taskModel/*[@taskid = $taskRef]"/>

  <td width="20%"> <!-- start of cell --><xsl:text>   </xsl:text>

    <!-- if this is a grouping task -->
    <xsl:if test="name($theTask) = 'groupingTask'">

      <!-- if it has a maximum, can't automate more than max tasks -->
      <xsl:if test="$theTask/@max != -1">
        <xsl:text>You may automate at most </xsl:text>
        <xsl:value-of select="$theTask/@max"/>
        <xsl:text> task(s).</xsl:text> <br/>
      </xsl:if>

      <!-- if it has a min then you must leave at least that many tasks -->
      <xsl:if test="$theTask/@min != -1">
        <xsl:text>You must leave at least </xsl:text>
        <xsl:value-of select="$theTask/@min"/>
        <xsl:text> task(s).</xsl:text>
      </xsl:if>

    </xsl:if>

  <!-- if this is an input task -->
    <xsl:if test="name($theTask) = 'inputTask'">

      <!-- if it has autogen, can create default info fields -->
      <xsl:if test="$theTask/autogen">
        <xsl:value-of select="$theTask/autogen/header"/><br /><br />
        <!-- Do fór each variable defined in autogen       -->
        <xsl:for-each select="$theTask/autogen/autoVar">
          <xsl:value-of select="./nameDisplay"/>
          <xsl:if test="./textDisplay">


            <xsl:element name="input">
              <xsl:attribute name="name">
                <xsl:text>defaults /defaultsContainer/</xsl:text>
                <xsl:value-of select="./@name"/>
                <!--<xsl:text>/defValue</xsl:text>-->
             <xsl:value-of select="./domainType"/>
                <xsl:text> </xsl:text>
              <xsl:value-of select="./varSequence"/>
              </xsl:attribute>
              <xsl:attribute name="size">
                <xsl:value-of select="./textDisplay/size"/>
              </xsl:attribute>
            </xsl:element>
          </xsl:if>
          <br />
        </xsl:for-each>

        <br />
        <xsl:value-of select="$theTask/autogen/footer"/>

      </xsl:if>

    </xsl:if>
  </td> <!-- end of cell -->

</xsl:template>


</xsl:stylesheet>
```

2. Profile Adaptation Stylesheet for Automating a Task

(from tamex/config/automateTask.xsl   11/21/02)

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- This stylesheet examines a task model and automates a task in it -->
<!-- The task whose id is supplied is automated. -->


<!-- this variable will hold the task id of the task to be automated -->
<xsl:param name="automate"/>

<xsl:template match="/">

  <!-- copy over the taskModel element as well as the namespace attributes -->
  <xsl:element name="taskModel">

    <!-- copy over the attributes be carefull howerver because the xmlns:xsi
         attribute is not copied, this is not a problem now because we are
         not validating the new task models but it should be looked at
         later -->
    <xsl:copy-of select="/taskModel/@*"/>

    <!-- copy over the domain element and the modelDescription element -->
    <xsl:copy-of select="/taskModel/domain"/>
    <xsl:copy-of select="/taskModel/modelDescription"/>

    <!-- for each grouping task see if you have a subtasks pointing
         to the task that will be automated then switch the activation
         attribute on those subtasks to auto -->
    <xsl:for-each select="/taskModel/groupingTask">
      <xsl:call-template name="groupingTaskFilter">
        <xsl:with-param name="gTask" select="./"/>
      </xsl:call-template>
    </xsl:for-each>

    <!-- copy over all of the other tasks -->
    <xsl:for-each select="/taskModel/*[(name() = ‘inputTask’) or (name() =
‘outputTask’) or (name() = ‘profileTask’) or (name() = ‘internalTask’) or
(name() = ‘wrapperTask’)]">

      <xsl:copy-of select="./"/>
    </xsl:for-each>
  </xsl:element> <!-- end taskModel element -->

</xsl:template>

<!-- Given a grouping task this template sets the tasks subtask task
     statement to auto activation if that particular subtask points
     to the task id given in the global var automate -->
<xsl:template name="groupingTaskFilter">
  <xsl:param name="gTask"/>

  <xsl:element name="groupingTask">
    <!-- copy over the attributes unchanged -->
    <xsl:copy-of select="$gTask/@*"/>
```

```
<!-- copy over the name,description functional constraints
     agent link and clear activation elements -->
<xsl:copy-of select="$gTask/name"/>
<xsl:copy-of select="$gTask/description"/>
<xsl:copy-of select="$gTask/functionalConstraints"/>
<xsl:copy-of select="$gTask/agent"/>
<xsl:copy-of select="$gTask/link"/>
<xsl:copy-of select="$gTask/clearActivation"/>

<!-- set the subtask task stmt to automatic if it's taskref attribute
     matches the value in the $automate variable -->
<xsl:for-each select="$gTask/subtask">

  <!-- if it does not match then just copy over the subtask -->
  <xsl:if test="not(./@taskref = $automate/value)">
    <xsl:copy-of select="./"/>
  </xsl:if>


  <!-- if it does match then set the activation attribute to auto -->
  <xsl:if test="./@taskref = $automate/value">
   <xsl:element name="subtask">
      <xsl:copy-of select="./@taskref"/>
      <xsl:copy-of select="./@name"/>
      <xsl:copy-of select="./@sequence"/>
      <xsl:copy-of select="./@required"/>
     <xsl:attribute
name="activation"><xsl:text>auto</xsl:text></xsl:attribute>
    </xsl:element>
  </xsl:if>

 </xsl:for-each>
</xsl:element> <!-- end taskModel element -->

</xsl:template>


</xsl:stylesheet>
```

3. Profile Adaptation Stylesheet for Removing a Task

(from tamex/config/removeTask.xsl 11/21/02)

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- This stylesheet examines a task model and removes a task from it -->
<!-- The task is removed by the id that is supplied, all subtask
     task statements that point to the deleated task are also deleted -->

<!-- this variable will hold the task id of the task to be removed -->
<xsl:param name="remove"/>

<xsl:template match="/">

  <!-- copy over the taskModel element as well as the namespace attributes -->
  <xsl:element name="taskModel">

    <!-- copy over the attributes be carefull howerver because the xmlns:xsi
         attribute is not copied, this is not a problem now because we are
         not validating the new task models -->
    <xsl:copy-of select="/taskModel/@*"/>

    <!-- copy over the domain element and the modelDescription element -->
    <xsl:copy-of select="/taskModel/domain"/>
    <xsl:copy-of select="/taskModel/modelDescription"/>

    <!-- copy over all of the grouping tasks taking out offending -->
    <!-- subtask elements, omit the grouping task if it's id matches -->
    <!-- the one that we are removing -->
    <xsl:for-each select="/taskModel/groupingTask">
      <xsl:call-template name="groupingTaskFilter">
        <xsl:with-param name="gTask" select="./"/>
      </xsl:call-template>
    </xsl:for-each>

    <!-- copy over all of the other tasks except for the one to be removed -->
    <xsl:for-each select="/taskModel/*[(name() = 'inputTask') or (name() =
'outputTask') or (name() = 'profileTask') or (name() = 'internalTask') or
(name() = 'wrapperTask')]">

      <xsl:call-template name="aTaskFilter">
        <xsl:with-param name="aTask" select="./"/>
      </xsl:call-template>

    </xsl:for-each>
  </xsl:element>

</xsl:template>

<!-- This template is passed a grouping task and filters out
     of this grouping task all subtask task statements that
     reference the task to be removed. Every thing else
     is left as is. -->
<xsl:template name="groupingTaskFilter">
  <xsl:param name="gTask"/>
  <xsl:if test="not($gTask/@taskid = $remove/value)">
    <xsl:element name="groupingTask">
      <!-- copy over the attributes unchanged -->
      <xsl:copy-of select="$gTask/@*"/>

      <!-- copy over the name,description functional constraints
           agent link and clear activation elements -->
      <xsl:copy-of select="$gTask/name"/>
      <xsl:copy-of select="$gTask/description"/>
      <xsl:copy-of select="$gTask/functionalConstraints"/>
      <xsl:copy-of select="$gTask/agent"/>
      <xsl:copy-of select="$gTask/link"/>
      <xsl:copy-of select="$gTask/clearActivation"/>

      <!-- copy over the subtask element if it does not point to task that we
           are removing -->
      <xsl:for-each select="$gTask/subtask">
        <xsl:if test="not(./@taskref = $remove/value)">
          <xsl:copy-of select="./"/>
        </xsl:if>
      </xsl:for-each>
    </xsl:element>
  </xsl:if>
</xsl:template>


<!-- This template will echo any task that it gets as an argument
     with the exception of a task that matches the task id of the
     task that we are to remove -->
<xsl:template name="aTaskFilter">
  <xsl:param name="aTask"/>

  <xsl:if test="not($aTask/@taskid = $remove/value)">
    <xsl:copy-of select="$aTask"/>
  </xsl:if>

</xsl:template>


</xsl:stylesheet>
```

D - System Configuration

1. Main Configuration - XML Schema

(from tamex/config/mainConfig.xsd 10/24/02)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd='http://www.w3.org/2000/10/XMLSchema'>

<xsd:element name="mainConfig">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="servletName" minOccurs = "1" maxOccurs = "1"/>
      <xsd:element ref="staticIntro" minOccurs = "0" maxOccurs = "1"/>
      <xsd:element ref="newUserIntro" minOccurs = "0" maxOccurs = "1"/>
      <xsd:element ref="mainDisplayXSL" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="agentPort" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="startAgent" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="strictValidationXSL" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="registeredTaskModels" minOccurs = "1" maxOccurs = "1"/>
      <xsd:element ref="registeredResources" minOccurs = "1" maxOccurs = "1"/>
      <xsd:element ref="registeredAgents" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="registeredTaskModels">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="model" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="model">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="modelName" minOccurs = "1" maxOccurs = "1"/>
      <xsd:element ref="path" minOccurs = "1" maxOccurs = "1"/>
    </xsd:sequence>
    <xsd:attribute name="validation" type="validationType" use="optional"
default="normal"/>
  </xsd:complexType>
</xsd:element>

<xsd:simpleType name = "validationType">
  <xsd:restriction base = "xsd:string">
    <xsd:enumeration value = "normal"/>
    <xsd:enumeration value = "strict"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="registeredResources">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="resource" minOccurs = "1" maxOccurs = "unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="resource">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="resourceName" minOccurs = "1" maxOccurs = "1"/>
      <xsd:element ref="requestTemplate" minOccurs = "1" maxOccurs = "1"/>
      <xsd:element ref="extractionGrammar" minOccurs = "1" maxOccurs = "1"/>
      <xsd:element ref="translationFile" minOccurs = "1" maxOccurs = "1"/>
      <xsd:element ref="cookies" minOccurs = "1" maxOccurs = "1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="registeredAgents">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="agentInfo" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="agentInfo">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="agentId" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="agentName" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="address" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="port" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="agentPort">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="startAgent">
  <xsd:simpleType>
    <xsd:restriction base="xsd:boolean">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>


<xsd:element name="address">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="agentName">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
```

```
</xsd:element>

<xsd:element name="agentId">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="port">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="servletName">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="staticIntro">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="newUserIntro">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="mainDisplayXSL">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="strictValidationXSL">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="modelName">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="path">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="resourceName">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="requestTemplate">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="extractionGrammar">
  <xsd:simpleType>
    <xsd:restriction base="nonEmptyString">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="translationFile">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="cookies">
  <xsd:simpleType>
    <xsd:restriction base="myBoolean">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:simpleType name="nonEmptyString">
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="1"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myBoolean">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="yes"/>
    <xsd:enumeration value="Yes"/>
    <xsd:enumeration value="no"/>
    <xsd:enumeration value="No"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

2. Main Configuration Instance - XML Schema Instance

(from tamex/config/mainConfig.xml   02/14/03)

```
<?xml version="1.0"?>
<mainConfig xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation='../../../config/mainConfig.xsd'>

 <!-- Name of the servlet -->
 <servletName>tamex.mediator.Mediator</servletName>

 <!-- Static part of the intro screen location -->
 <staticIntro>../../../config/staticIntro.html</staticIntro>

 <!-- Instructions displayed to a user when first logs on -->
 <newUserIntro>../../../config/newUserIntro.html</newUserIntro>

 <!-- location of the main display stylesheet -->
 <mainDisplayXSL>../../../config/mainDisplay.xsl</mainDisplayXSL>

 <!-- this port number is the one that the AgentServer will run on -->
 <agentPort>2222</agentPort>

 <!-- this tells the mediator wether or not to activate agent services -->
 <startAgent>false</startAgent>

 <!-- specifys the xsl file that will do strict validation of task models -->

<strictValidationXSL>../../../config/strictValidation.xsl</strictValidationXSL>

 <!-- List of the registered task models -->
 <registeredTaskModels>
  <model validation="normal">
    <modelName>myBookBuying Small</modelName>
    <path>../../../taskModels/myBookBuying/BookBuyingSmall.xml</path>
  </model>
<!--
  <model validation="normal">
    <modelName>test1</modelName>
    <path>../../../taskModels/test1/test1TaskModel.xml</path>
  </model>

  <model>
    <modelName>test4</modelName>
    <path>../../../taskModels/test4/testTaskModel4.xml</path>
  </model>

  <model>
    <modelName>BookBuying</modelName>
    <path>../../../taskModels/BookBuying/BookBuying.xml</path>
  </model>
-->
  <model validation="normal">
    <modelName>myBookBuying</modelName>
    <path>../../../taskModels/myBookBuying/BookBuying.xml</path>
  </model>
```

```
    <model>
      <modelName>Pharmacy Shopping</modelName>
      <path>../../../taskModels/PharmacyShopping/PharmacyShopping.xml</path>
    </model>

    <model>
      <modelName>Surgery 446</modelName>
      <path>../../../taskModels/Surgery/Surgery.xml</path>
    </model>

    <model>
      <modelName>Surgery 446 PalmOS</modelName>
      <path>../../../taskModels/Surgery/SurgeryPalm.xml</path>
    </model>

  <!--
    <model>
      <modelName>Search</modelName>
      <path>../../../taskModels/Search/Search.xml</path>
    </model>

    <model>
      <modelName>Quotes</modelName>
      <path>../../../taskModels/Quotes/Quotes.xml</path>
    </model>

    <model>
      <modelName>TravelPlanning</modelName>
      <path>../../../taskModels/TravelPlanning/TravelPlanning.xml</path>
    </model>
  -->

   </registeredTaskModels>

   <!-- List of all the resources that the mediator knows about -->
   <registeredResources>
  <!--   <resource>
       <resourceName>Yahoo</resourceName>
       <requestTemplate>../../../wrappers/yahoo-
finance/RequestProtocols/protocol.xml</requestTemplate>
       <extractionGrammar>../../../wrappers/yahoo-
finance/LearnedResponseGrammar/grammar.xml</extractionGrammar>
       <translationFile></translationFile>
       <cookies>yes</cookies>
     </resource>

     <resource>
       <resourceName>etrade</resourceName>

<requestTemplate>../../../wrappers/etrade/RequestProtocols/protocol.xml</reques
tTemplate>

<extractionGrammar>../../../wrappers/etrade/LearnedResponseGrammar/grammar.xml<
/extractionGrammar>
       <translationFile></translationFile>
       <cookies>yes</cookies>
     </resource>

     <resource>
       <resourceName>quote</resourceName>
```

```
<requestTemplate>../../../wrappers/quote/RequestProtocols/protocol.xml</request
Template>

<extractionGrammar>../../../wrappers/quote/LearnedResponseGrammar/grammar.xml</
extractionGrammar>
     <translationFile></translationFile>
     <cookies>yes</cookies>
   </resource>

   <resource>
     <resourceName>LowestFare</resourceName>

<requestTemplate>../../../wrappers/lowestfare/RequestProtocols/protocol1.xml</r
equestTemplate>

<extractionGrammar>../../../wrappers/lowestfare/LearnedResponseGrammar/grammar.
xml</extractionGrammar>
     <translationFile></translationFile>
     <cookies>yes</cookies>
   </resource>

   <resource>
     <resourceName>YahooSearch</resourceName>

<requestTemplate>../../../wrappers/yahoo/RequestProtocols/protocol.xml</request
Template>

<extractionGrammar>../../../wrappers/yahoo/LearnedResponseGrammar/grammar.xml</
extractionGrammar>
     <translationFile></translationFile>
     <cookies>yes</cookies>
   </resource>

   <resource>
     <resourceName>Google</resourceName>

<requestTemplate>../../../wrappers/google/RequestProtocols/protocol.xml</reques
tTemplate>

<extractionGrammar>../../../wrappers/google/LearnedResponseGrammar/grammar.xml<
/extractionGrammar>
     <translationFile></translationFile>
     <cookies>yes</cookies>
   </resource>-->

   <resource>
     <resourceName>book-1book</resourceName>
     <requestTemplate>../../../wrappers/book-
1book/RequestProtocols/protocol.xml</requestTemplate>
     <extractionGrammar>../../../wrappers/book-
1book/LearnedResponseGrammar/grammar1.xml</extractionGrammar>
     <translationFile></translationFile>
     <cookies>yes</cookies>
   </resource>

   <resource>
     <resourceName>book-amazon</resourceName>
     <requestTemplate>../../../wrappers/book-
amazon/RequestProtocols/protocol.xml</requestTemplate>
```

```
      <extractionGrammar>../../../wrappers/book-
amazon/LearnedResponseGrammar/grammar1.xml</extractionGrammar>
      <translationFile></translationFile>
      <cookies>yes</cookies>
    </resource>

    <resource>
      <resourceName>book-chapters</resourceName>
      <requestTemplate>../../../wrappers/book-
chapters/RequestProtocols/protocol.xml</requestTemplate>
      <extractionGrammar>../../../wrappers/book-
chapters/LearnedResponseGrammar/grammar1.xml</extractionGrammar>
      <translationFile></translationFile>
      <cookies>yes</cookies>
    </resource>

    <resource>
      <resourceName>drugs-emedical</resourceName>
      <requestTemplate>../../../wrappers/drugs-
emedical/RequestProtocols/protocol.xml</requestTemplate>
      <extractionGrammar>../../../wrappers/drugs-
emedical/LearnedResponseGrammar/grammar.xml</extractionGrammar>
      <translationFile></translationFile>
      <cookies>yes</cookies>
    </resource>

    <resource>
      <resourceName>drugs-pdirect</resourceName>
      <requestTemplate>../../../wrappers/drugs-
pdirect/RequestProtocols/protocol.xml</requestTemplate>
      <extractionGrammar>../../../wrappers/drugs-
pdirect/LearnedResponseGrammar/grammar.xml</extractionGrammar>
      <translationFile></translationFile>
      <cookies>yes</cookies>
    </resource>

    <resource>
      <resourceName>drugs-drugstore</resourceName>
      <requestTemplate>../../../wrappers/drugs-
drugstore/RequestProtocols/protocol.xml</requestTemplate>
      <extractionGrammar>../../../wrappers/drugs-
drugstore/LearnedResponseGrammar/grammar.xml</extractionGrammar>
      <translationFile></translationFile>
      <cookies>yes</cookies>
    </resource>

    <resource>
      <resourceName>medterms</resourceName>

<requestTemplate>../../../wrappers/medterms/RequestProtocols/protocol.xml</requ
estTemplate>

<extractionGrammar>../../../wrappers/medterms/LearnedResponseGrammar/grammar1.x
ml</extractionGrammar>
      <translationFile></translationFile>
      <cookies>yes</cookies>
    </resource>

    <resource>
      <resourceName>medterms2</resourceName>
```

```
<requestTemplate>../../../wrappers/medterms2/RequestProtocols/protocol.xml</req
uestTemplate>

<extractionGrammar>../../../wrappers/medterms2/LearnedResponseGrammar/grammar1.
xml</extractionGrammar>
     <translationFile></translationFile>
     <cookies>yes</cookies>
   </resource>

 </registeredResources>

<!-- List of the registered agents, referenced by agentId in the task model-->
 <registeredAgents>
   <agentInfo>
     <agentId>A01</agentId>
     <agentName>Test agent 1</agentName>
     <address>stauffer.cs.ualberta.ca</address>
     <port>2222</port>
   </agentInfo>
   <agentInfo>
     <agentId>A02</agentId>
     <agentName>Test agent 2</agentName>
     <address>erskine.cs.ualberta.ca</address>
     <port>2222</port>
   </agentInfo>
   <agentInfo>
     <agentId>eddie01</agentId>
     <agentName>eddies test agent 1</agentName>
     <address>peerless.cs.ualberta.ca</address>
     <port>2222</port>
   </agentInfo>

 </registeredAgents>


</mainConfig>
```

E - User Interface Configuration

1. Main Display - XML Schema

(from tamex/config/mainDisplay.xsd   09/29/02)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd='http://www.w3.org/2000/10/XMLSchema'>

<!-- This is the element that will be generated by each call
     to the mediator. It can now hold an intro screen or a normal
     screen, but not both. This will be used as the last level of
     xsl application before the user sees the data. -->
<xsd:element name="mainDisplay">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="servletName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
      <xsd:element ref="pageTitle" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="currentTmName" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:choice>
        <xsd:element ref="introScreen" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="normalPage" minOccurs="1" maxOccurs="1"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>


<!-- This element describes the intro screen produced when a user
     first accesses the mediator -->
<xsd:element name="introScreen">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="staticIntro" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="taskModel" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- This element contains the static intro information found in the
     static intro configuration file. -->
<xsd:element name="staticIntro">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="skip"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- This element describes info about a task model, namely it's name
     and a description of what it does -->
<xsd:element name="taskModel">
  <xsd:complexType>
    <xsd:sequence>
```

```
      <xsd:element name="tmName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="tmDesc" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- this describes the screens produced through most of the user's
     interaction witht he mediator. Other than the intro screen
     this is going to be the xml that will be generated. -->
<xsd:element name="normalPage">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="navTree" minOccurs="0" maxOccurs = "1"/>
      <xsd:element ref="taskData" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="errors" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- This element contains all of the information needed to create
     a navigation tree for the user. -->
<xsd:element name="navTree">
  <xsd:complexType>
    <xsd:sequence>
      <!-- there can only be one root element in a task model -->
      <xsd:element ref="taskInfo" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- This element specifys all of the data needed to identify and properly
     display an individual task to the user, this also includes a list
     of subtasks (if this is a grouping task) -->
<xsd:element name="taskInfo">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="taskId" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="taskName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="canExec" type="xsd:boolean" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="complete" type="xsd:boolean" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="started" type="xsd:boolean" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="current" type="xsd:boolean" minOccurs="1"
maxOccurs="1"/>
      <xsd:element ref="taskInfo" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>


<!-- This is the element that holds the data returned by the task
     execution, this will most likely be html code that will just
     be plugged into the display page. -->
<xsd:element name="taskData">
```

```
  <xsd:complexType>
    <xsd:sequence>
      <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="skip"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="errors">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="error" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="error">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="type" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="desc" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="advice" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- the tile of the page to be displayed -->
<xsd:element name="pageTitle" type="xsd:string" />

</xsd:schema>
```

2. Main Display - XSL Stylesheet for Generating User Interface

(from tamex/config/mainDisplay.xsl 03/31/03)

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<HTML>
<HEAD>
<TITLE><xsl:value-of select="/mainDisplay/pageTitle"/></TITLE>
</HEAD>
<BODY>

<!-- If this is an intro screen -->
<xsl:if test="not(string-length(/mainDisplay/introScreen) = 0)">
  <!-- pump out the static intro information -->
  <xsl:copy-of select="/mainDisplay/introScreen/staticIntro"/>

  <!-- table to hold the task models -->
  <table width="100%" cols="2">
    <!-- Create the headings. -->
    <tr><td width="15%"><b>TaskModel Name:</b><hr width="100%"/></td>
        <td width="85%"><b>TaskModel Description:</b><hr
width="100%"/></td></tr>

    <!-- for each supported task model create a link -->
    <xsl:for-each select="/mainDisplay/introScreen/taskModel">
      <tr>
      <td width="15%" valign="top"> <!-- place for name of task model -->
        <xsl:element name="a">
         <xsl:attribute name="href">
           <xsl:value-of
select="/mainDisplay/servletName"/>?modelName=<xsl:value-of select="./tmName"/>
         </xsl:attribute>
         <b><xsl:value-of select="./tmName"/>:</b>
        </xsl:element>
        <hr width="100%"/>
      </td>
      <td width="85%"> <!-- place for desc of task model -->
        <xsl:value-of select="./tmDesc"/>
        <hr width="100%"/>
      </td>
      </tr>
    </xsl:for-each>
  </table>

</xsl:if>
<!-- end intro screen -->


<!-- If this is a normal interaction screen -->
<xsl:if test="not(string-length(/mainDisplay/normalPage) = 0)">
  <!-- begin main table where nav tree and data pages go -->
  <table width="100%" border="true" cols="2">
   <tr>
   <td width="30%" valign="top"> <!-- navigation tree -->
    <xsl:apply-templates select="/mainDisplay/normalPage/navTree/taskInfo"/>
    <xsl:call-template name="addButtons"/>
   </td>
   <td width="70%" valign="top"> <!-- data for the user -->
     <xsl:copy-of select="/mainDisplay/normalPage/taskData"/>
   </td>
   </tr>
   <!-- add error data if any -->
   <tr>
    <span><xsl:apply-templates select="/mainDisplay/normalPage/errors"/></span>
   </tr>
  </table> <!-- end main table -->
</xsl:if>
<!-- end normal interaction screen -->


</BODY>
</HTML>
</xsl:template>

<!-- This template makes the links and draws the navigation tree
     using a list. -->
<xsl:template match="taskInfo">
  <ul>
  <li>  <!-- The color coded link goes here -->

   <!-- add the link for the description -->
   <xsl:element name="a">
     <xsl:attribute name="href"><xsl:value-of
select="/mainDisplay/servletName"/>?taskId=<xsl:value-of
select="./taskId"/>&amp;desc ription=true</xsl:attribute>
D
   </xsl:element>

   <!-- link -->
   <xsl:element name="a">
     <xsl:attribute name="href">
       <xsl:value-of select="/mainDisplay/servletName"/>?taskId=<xsl:value-of
select="./taskId"/></xsl:attribute>

     <!-- text description of link -->
     <xsl:if test="./complete='true'">
       <font color="blue" size="-1">
         <xsl:value-of select="./taskName"/>
         (<xsl:value-of select="./taskId"/>)
       </font>
     </xsl:if>

     <xsl:if test="(./complete='false') and (./canExec='true')">
       <font color="green" size="-1">
         <xsl:value-of select="./taskName"/>
         (<xsl:value-of select="./taskId"/>)
       </font>
     </xsl:if>

     <xsl:if test="(./complete='false') and (./canExec='false')">
       <font color="red" size="-1">
         <xsl:value-of select="./taskName"/>
         (<xsl:value-of select="./taskId"/>)
       </font>
     </xsl:if>
   <!-- end link -->
   </xsl:element>


   <!-- if current add "<==" -->
   <xsl:if test="./current='true'">
&lt;==
   </xsl:if>

  </li>

  <!-- recurse for others if this is a grouping task -->
  <xsl:for-each select="./taskInfo">
    <xsl:apply-templates select="."/>
  </xsl:for-each>
  </ul>
</xsl:template>

<!-- This template formats the errors if any -->
<xsl:template match="errors">

<center><H3>Unrecoverable Errors</H3></center>
<table width="100%">
 <tr valign="top">
   <td width="15%"><b>Error Name:</b></td>
   <td width="85%"><xsl:value-of select="./error/name"/></td>
 </tr>
 <tr valign="top">
   <td width="15%"><b>Error Type:</b></td>
   <td width="85%"><xsl:value-of select="./error/type"/></td>
 </tr>
 <tr valign="top">
   <td width="15%"><b>Error Description:</b></td>
   <td width="85%"><xsl:value-of select="./error/desc"/></td>
 </tr>
 <tr valign="top">
   <td width="15%"><b>Advice:</b></td>
   <td width="85%"><xsl:value-of select="./error/advice"/></td>
 </tr>
 <tr valign="top">
   <td width="15%"> </td>
   <td width="85%"><hr width="100%"/></td>
 </tr>
</table>

</xsl:template>

<!-- this template adds the navigation buttons -->
<xsl:template name="addButtons">
  <hr width="100%"/>
  <center><font size="-1">General Navigation Options:</font></center>
  <table width="100%" cols="2">
    <tr>
      <td valign="top" width="45%"> <!-- add the main button -->
        <xsl:element name="a">
          <xsl:attribute name="href">
            <xsl:value-of select="/mainDisplay/servletName"/>
          </xsl:attribute>
            <font size="-1">Go To Main / Log Out</font>
        </xsl:element>
      </td>
      <td valign="top" width="55%">
        <xsl:element name="a"> <!-- restart task model -->
          <xsl:attribute name="href">
            <xsl:value-of
select="/mainDisplay/servletName"/>?modelName=<xsl:value-of
select="/mainDisplay/currentTmName"/>
          </xsl:attribute>
            <font size="-1">Restart this task model</font>
        </xsl:element>
      </td>
      </tr>
      <tr>
      <td valign="top" width="35%">
        <xsl:element name="a">
          <xsl:attribute name="href">
            <xsl:text>../help.html</xsl:text>
          </xsl:attribute>
            <font size="-1">Help</font>
        </xsl:element>
      </td>
      <td valign="top" width="65%">
        <xsl:element name="a">
          <xsl:attribute name="href">
            <xsl:text>../about.html</xsl:text>
          </xsl:attribute>
            <font size="-1">About</font>
        </xsl:element>
      </td>
    </tr>
  </table>
  <hr width="100%"/>
</xsl:template>

</xsl:stylesheet>
```

F - Validation

1. Strict Validation Stylesheet

(from tamex/config/strictValidation.xsl 08/19/02)

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xalan"
exclude-result-prefixes="xalan">

<!-- The main config file optionally passed in to check if resources
     used in this task model are indeed there, also used to check
     if agent id's used have been defined -->
<xsl:param name="config"/>

<xsl:variable name="root" select="/"/>

<xsl:template match="/">

  <!-- check that task ids are unique -->
  <xsl:call-template name="checkTaskIdUniqueness"/>

  <!-- check that task ids are valid format -->
  <xsl:call-template name="checkTIdValidity"/>

  <!-- check that task statement ids are valid  -->
  <xsl:call-template name="checkTSIdValidity"/>

  <!-- check that the start attribute points at a valid task id -->
  <xsl:call-template name="checkStart"/>

  <!-- check that the subtasks point at valid task ids -->
  <xsl:call-template name="checkSubtasks"/>

  <!-- check that the links point at valid task ids -->
  <xsl:call-template name="checkLinks"/>

  <!-- check that the clear activation elements point at valid task ids -->
  <xsl:call-template name="checkClearActivation"/>

  <!-- variable and info in elements may not contain the " char -->
  <xsl:call-template name="doVarsHaveQuotes"/>

  <!-- check variable names are unique in each var element of a task -->
  <xsl:call-template name="checkVarNameUniqueness"/>

  <!-- check that variable names are unique in each infoIn statement of a task
-->
  <xsl:call-template name="checkInfoInNameUniqueness"/>

  <!-- for each var defined, it's name may not be used inside an infoIn element
-->
  <xsl:call-template name="checkVarVsInfoName"/>

  <!-- for each use of a variable check if that variable has been defined -->
  <xsl:call-template name="areVariablesDefined"/>
  <!-- if the main Config file has been supplied -->
  <xsl:if test="$config != ''">
    <!-- for each wrapper call check if that wrapper has been defined in main
config -->
    <xsl:call-template name="checkWrappers"/>

    <!-- for each agent reference check if it has been defined -->
    <xsl:call-template name="checkAgents"/>
  </xsl:if>


</xsl:template>


<!-- Checks to see that all of the defined tasks have unique task ids -->
<xsl:template name="checkTaskIdUniqueness">

  <!-- add all task ids in "" seperated string -->
  <xsl:variable name="all">
    <xsl:for-each select="$root//@taskid">
      <xsl:text>"</xsl:text><xsl:value-of select="./"/><xsl:text>"</xsl:text>
    </xsl:for-each>
  </xsl:variable>

  <xsl:for-each select="$root//@taskid">

    <xsl:variable name="taskId">
      <xsl:text>"</xsl:text><xsl:value-of select="./"/><xsl:text>"</xsl:text>
    </xsl:variable>

    <xsl:variable name="count">
      <xsl:call-template name="count-substrings">
        <xsl:with-param name="text" select="$all"/>
        <xsl:with-param name="substring" select="$taskId"/>
      </xsl:call-template>
    </xsl:variable>

    <xsl:if test="number($count) &gt; 1">
      <xsl:text>Error: The task id "</xsl:text>
      <xsl:value-of select="./"/>
      <xsl:text>" is used </xsl:text>
      <xsl:value-of select="$count"/>
      <xsl:text> times but task ids must be unique.&#10;&#10;</xsl:text>
    </xsl:if>

  </xsl:for-each>

</xsl:template>

<!-- Checks to make sure that task ids do not contain quotes or
     pipe chars. -->
<xsl:template name="checkTIdValidity">

  <!-- first check the start attribute -->
  <xsl:variable name="startId" select="/taskModel/@start"/>

  <xsl:if test="contains($startId,'|')">
    <xsl:text>Error: The start attribute of the "</xsl:text>
    <xsl:value-of select="/taskModel/@name"/>
    <xsl:text>" contains the | character, which is not allowed in task
ids.</xsl:text>
    <xsl:text>&#10;&#10;</xsl:text>
  </xsl:if>

  <xsl:if test="contains($startId,'&quot;')">
    <xsl:text>Error: The start attribute of the "</xsl:text>
    <xsl:value-of select="/taskModel/@name"/>
    <xsl:text>" contains the " character, which is not allowed in task
ids.</xsl:text>
    <xsl:text>&#10;&#10;</xsl:text>
  </xsl:if>

  <!-- check all of the task id's attached to tasks -->
  <xsl:for-each select="//@taskid">

    <xsl:if test="contains(./,'|')">
      <xsl:text>Error: The task id of task "</xsl:text>
      <xsl:value-of select="./"/>
      <xsl:text>" in task model "</xsl:text>
      <xsl:value-of select="/taskModel/@name"/>
      <xsl:text>" contains the | character, which is not allowed in task
ids.</xsl:text>
      <xsl:text>&#10;&#10;</xsl:text>
    </xsl:if>

    <xsl:if test="contains(./,'&quot;')">
      <xsl:text>Error: The task id of task "</xsl:text>
      <xsl:value-of select="./"/>
      <xsl:text>" in task model "</xsl:text>
      <xsl:value-of select="/taskModel/@name"/>
      <xsl:text>" contains the " character, which is not allowed in task
ids.</xsl:text>
      <xsl:text>&#10;&#10;</xsl:text>
    </xsl:if>
  </xsl:for-each>

</xsl:template>

<!-- Checks to make sure that task statement id's, ie externalId's do
     not contain quotes and are properly formed triplets. -->
<xsl:template name="checkTSIdValidity">

  <xsl:for-each select="//@externalId">
    <xsl:if test="contains(./,'&quot;')">
      <xsl:text>Error: The task statement id of task statement "</xsl:text>
      <xsl:value-of select="./"/>
      <xsl:text>" in task model "</xsl:text>
      <xsl:value-of select="/taskModel/@name"/>
      <xsl:text>" contains the " character, which is not allowed </xsl:text>
      <xsl:text>in task statement ids.</xsl:text>
      <xsl:text>&#10;&#10;</xsl:text>
    </xsl:if>

    <!-- grab all of the | delimited parts of the external id -->
    <xsl:variable name="parts">
      <xsl:call-template name="split-externalId">
        <xsl:with-param name="text" select="./"/>
      </xsl:call-template>
    </xsl:variable>
    <!-- count the parts it has to be a triplet -->
    <xsl:if test="count(xalan:nodeset($parts)/part) != 3">
      <xsl:text>Error: The externalId specified, "</xsl:text>
      <xsl:value-of select="./"/>
      <xsl:text>" in task model "</xsl:text>
      <xsl:value-of select="/taskModel/@name"/>
      <xsl:text>" is not a valid triplet.&#10;&#10;</xsl:text>
    </xsl:if>

    <xsl:variable name="externalId" select="./"/>

    <!-- check to see if all three parts are full and not just
         empty strings. -->
    <xsl:for-each select="xalan:nodeset($parts)/part">
      <xsl:if test="string-length(./) = 0">
        <xsl:text>Error: The externalId specified, "</xsl:text>
        <xsl:value-of select="$externalId"/>
        <xsl:text>" in task model "</xsl:text>
        <xsl:value-of select="$root/taskModel/@name"/>
        <xsl:text>" is not a valid triplet. Empty clauses are not
allowed.&#10;&#10;</xsl:text>
      </xsl:if>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>

<!-- Checks if the task identified as the start task actually exists -->
<xsl:template name="checkStart">

  <xsl:variable name="flag">
    <xsl:call-template name="isTaskIdDefined">
      <xsl:with-param name="id" select="$root/taskModel/@start"/>
      <xsl:with-param name="rootOfTm" select="$root"/>
    </xsl:call-template>
  </xsl:variable>

  <xsl:if test="not(number($flag))">
    <xsl:text>Error: The start attribute of the "</xsl;text>
    <xsl:value-of select="/taskModel/@name"/>
    <xsl:text>" task model points at a task that has not been
defined.&#10;&#10;</xsl:text>
  </xsl:if>
</xsl:template>

<!-- This template checks all of the subtask elements in the task model,
     it makes sure that the taskref attribute points to a defined task
     and that the name of that task matches the name of the subtask's
     name attribute -->
<xsl:template name="checkSubtasks">

  <xsl:for-each select="//subtask">

    <!-- check that the task referenced has been defined -->
    <xsl:variable name="flag">
      <xsl:call-template name="isTaskIdDefined">
        <xsl:with-param name="id" select="./@taskref"/>
        <xsl:with-param name="rootOfTm" select="$root"/>
      </xsl:call-template>
    </xsl:variable>

    <xsl:if test="not(number($flag))">
```

```
      <xsl:text>Error: The subtask task statement in the task "</xsl:text>
      <xsl:value-of select="../@taskid"/>
      <xsl:text>" from the "</xsl:text>
      <xsl:value-of select="$root/taskModel/@name"/>
      <xsl:text>" task model references a task with id "</xsl:text>
      <xsl:value-of select="./@taskref"/>
      <xsl:text>" but a task with that id has not been
defined.&#10;&#10;</xsl:text>
    </xsl:if>

    <!-- now check that the name attribute matches the name of the task -->
    <!-- first get the task with that id -->
    <xsl:variable name="taskRef" select="./@taskref"/>
    <xsl:variable name="task" select="$root//*[@taskid = $taskRef]"/>

    <xsl:if test="not(./@name = $task/name)">
      <xsl:text>Error: The name attribute provided in subtask task statement
specified inside a grouping task does not match the name element in the
referenced task.&#10;</xsl:text>
      <xsl:text>Detailes...&#10;</xsl:text>
      <xsl:text>Subtask declared in task "</xsl:text><xsl:value-of
select="../@taskid"/>
      <xsl:text>", references task with id "</xsl:text><xsl:value-of
select="./@taskref"/>
      <xsl:text>".&#10;</xsl:text>
      <xsl:text>Subtask name attribute "</xsl:text><xsl:value-of
select="./@name"/>
      <xsl:text>".&#10;</xsl:text>
      <xsl:text>Task name element      "</xsl:text><xsl:value-of
select="$task/name"/>
      <xsl:text>".&#10;&#10;</xsl:text>
    </xsl:if>
  </xsl:for-each>

</xsl:template>

<!-- This template makes sure that task ids specified inside
     link elements have been defined. -->
<xsl:template name="checkLinks">

  <!-- for each link element check if the id has been defined -->
  <xsl:for-each select="$root//link">

    <xsl:variable name="flag">
      <xsl:call-template name="isTaskIdDefined">
        <xsl:with-param name="id" select="./"/>
        <xsl:with-param name="rootOfTm" select="$root"/>
      </xsl:call-template>
    </xsl:variable>

    <xsl:if test="not(number($flag))">
      <xsl:text>Error: Task id referenced in link element has not been
defined.&#10;</xsl:text>
      <xsl:text>Detailes:&#10;</xsl:text>
      <xsl:text>Link element occurs inside task with id "</xsl:text>
      <xsl:value-of select="../@taskid"/>
      <xsl:text>".&#10;Link element references id "</xsl:text>
      <xsl:value-of select="./"/>
      <xsl:text>" which does not exist.&#10;&#10;</xsl:text>
    </xsl:if>
  </xsl:for-each>
</xsl:template>


<!-- This template makes sure that task ids specified inside
     clearActivation elements have been defined. -->
<xsl:template name="checkClearActivation">

  <!-- for each clearActivation element check if the id has been defined -->
  <xsl:for-each select="$root//clearActivation">

    <xsl:variable name="flag">
      <xsl:call-template name="isTaskIdDefined">
        <xsl:with-param name="id" select="./"/>
        <xsl:with-param name="rootOfTm" select="$root"/>
      </xsl:call-template>
    </xsl:variable>

    <xsl:if test="not(number($flag))">
      <xsl:text>Error: Task id referenced in clearActivation element has not
been defined.&#10;</xsl:text>
      <xsl:text>Detailes:&#10;</xsl:text>
      <xsl:text>ClearActivation element occurs inside task with id "</xsl:text>
      <xsl:value-of select="../@taskid"/>
      <xsl:text>".&#10;ClearActivation element references id "</xsl:text>
      <xsl:value-of select="./"/>
      <xsl:text>" which does not exist.&#10;&#10;</xsl:text>
    </xsl:if>

  </xsl:for-each>
</xsl:template>

<!-- This template makes sure that no variables have quote characters in them -
->
<xsl:template name="doVarsHaveQuotes">

  <!-- get the set of var name and infoIn/infoName elements -->
  <xsl:for-each select="$root//var/@name|$root//infoIn/infoName">

    <xsl:if test="contains(./,‘&quot;’)">
      <xsl:text>Error: Variable or InfoIn element name contains a &quot;
character.</xsl:text>
      <xsl:text> The name in question is "</xsl:text>
      <xsl:value-of select="./"/>
      <xsl:text>".&#10;&#10;</xsl:text>
    </xsl:if>

  </xsl:for-each>
</xsl:template>

<!-- This template goes through each task and checks that there is no
     var element that specifys the same name attribute -->
<xsl:template name="checkVarNameUniqueness">

  <!-- for each task in the task model -->
  <xsl:for-each select="$root//@taskid">
    <!-- get the task that the id belongs to -->
    <xsl:variable name="task" select="../"/>

    <!-- get all var names in a " seperated list -->
    <xsl:variable name="all">
```

```
      <xsl:for-each select="$task//var">
        <xsl:text>"</xsl:text><xsl:value-of
select="@name"/><xsl:text>"</xsl:text>
      </xsl:for-each>
    </xsl:variable>

    <!-- each variable should only occur once -->
    <xsl:for-each select="$task/var">

      <xsl:variable name="count">
        <xsl:call-template name="count-substrings">
          <xsl:with-param name="text" select="$all"/>
          <xsl:with-param name="substring"
select="concat('&quot;',./@name,'&quot;')"/>
        </xsl:call-template>
      </xsl:variable>

      <xsl:if test="number($count) &gt; 1">
        <xsl:text>Error: The variable name "</xsl:text>
        <xsl:value-of select="./@name"/>
        <xsl:text>" is used </xsl:text>
        <xsl:value-of select="$count"/>
        <xsl:text> times inside task "</xsl:text>
        <xsl:value-of select="$task/@taskid"/>
        <xsl:text>"&#10;&#10;</xsl:text>
      </xsl:if>

    </xsl:for-each>
  </xsl:for-each>
</xsl:template>

<!-- This template goes through each task and checks that there is no
     infoIn element that specifys the same infoName element -->
<xsl:template name="checkInfoInNameUniqueness">

  <!-- for each task in the task model -->
  <xsl:for-each select="$root//@taskid">
    <!-- get the task that the id belongs to -->
    <xsl:variable name="task" select="../"/>

    <!-- get all infoIn names in a " seperated list -->
    <xsl:variable name="all">
      <xsl:for-each select="$task//infoIn/infoName">
        <xsl:text>"</xsl:text><xsl:value-of select="./"/><xsl:text>"</xsl:text>
      </xsl:for-each>
    </xsl:variable>

    <!-- each variable should only occur once -->
    <xsl:for-each select="$task//infoIn/infoName">

      <xsl:variable name="count">
        <xsl:call-template name="count-substrings">
          <xsl:with-param name="text" select="$all"/>
          <xsl:with-param name="substring"
select="concat('&quot;',./,'&quot;')"/>
        </xsl:call-template>
      </xsl:variable>

      <xsl:if test="number($count) &gt; 1">
        <xsl:text>Error: The info name "</xsl:text>
        <xsl:value-of select="./"/>
        <xsl:text>" is used </xsl:text>
        <xsl:value-of select="$count"/>
        <xsl:text> times inside task "</xsl:text>
        <xsl:value-of select="$task/@taskid"/>
        <xsl:text>"&#10;&#10;</xsl:text>
      </xsl:if>

    </xsl:for-each>

  </xsl:for-each>
</xsl:template>

<!-- This template makes sure that any variable name defined using a var
     element does not apper inside an infoName element of the same task. -->
<xsl:template name="checkVarVsInfoName">

  <!-- for each task in the task model -->
  <xsl:for-each select="$root//@taskid">
    <!-- get the task that the id belongs to -->
    <xsl:variable name="task" select="../"/>

    <!-- get all infoIn names in a " seperated list -->
    <xsl:variable name="all">
      <xsl:for-each select="$task//infoIn/infoName">
        <xsl:text>"</xsl:text><xsl:value-of select="./"/><xsl:text>"</xsl:text>
      </xsl:for-each>
    </xsl:variable>

    <!-- each variable should not have it's name in the info in statement -->
    <xsl:for-each select="$task//var">

      <xsl:variable name="count">
        <xsl:call-template name="count-substrings">
          <xsl:with-param name="text" select="$all"/>
          <xsl:with-param name="substring"
select="concat('&quot;',./@name,'&quot;')"/>
        </xsl:call-template>
      </xsl:variable>

      <xsl:if test="number($count) &gt;= 1">
        <xsl:text>Error: The variable name "</xsl:text>
        <xsl:value-of select="./@name"/>
        <xsl:text>" is also used in an infoIn element. Variable names which
have already been defined may not be used inside the infoIn task statement.
</xsl:text>
        <xsl:text>Error occured inside task "</xsl:text>
        <xsl:value-of select="$task/@taskid"/>
        <xsl:text>"&#10;&#10;</xsl:text>
      </xsl:if>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>

<!-- This template verifys that variables have been defined before they are
used -->
<!-- For each task there will be a section of task statements that need to be
checked
     it is this list that needs to be expanded when adding task statements
which
     use variables. -->
<xsl:template name="areVariablesDefined">
```

```
  <!-- for each task perform the check -->
  <xsl:for-each select="//@taskid">

    <!-- grab the task element -->
    <xsl:variable name="task" select="../"/>

    <!-- grab a list of the defined variable names -->
    <xsl:variable name="defined">
      <xsl:for-each select="$task/var/@name|$task//infoIn/infoName">
        <xsl:text>"</xsl:text><xsl:value-of select="./"/><xsl:text>"</xsl:text>
      </xsl:for-each>
    </xsl:variable>

    <!-- check var usage in xslt element, if there is one -->
    <xsl:if test="$task/xslt">

      <!-- check the in elements -->
      <xsl:for-each select="$task/xslt/in">
        <xsl:if test="./@var = ’true’">
          <xsl:if test="not(contains($defined,concat(’&quot;’,./,’&quot;’)))">
            <xsl:text>Error: Variable "</xsl:text>
            <xsl:value-of select="./"/>
            <xsl:text>" referenced inside xslt/in task statement in task
"</xsl:text>
            <xsl:value-of select="$task/@taskid"/>
            <xsl:text>" has not been defined.&#10;&#10;</xsl:text>
          </xsl:if>
        </xsl:if>
      </xsl:for-each>

      <!-- check the xsl element -->
      <xsl:if test="$task/xslt/xsl/@var = ’true’">
        <xsl:if
test="not(contains($defined,concat(’&quot;’,$task/xslt/xsl,’&quot;’)))">
          <xsl:text>Error: Variable "</xsl:text>
          <xsl:value-of select="$task/xslt/xsl"/>
          <xsl:text>" referenced inside xslt/xsl task statement in task
"</xsl:text>
          <xsl:value-of select="$task/@taskid"/>
          <xsl:text>" has not been defined.&#10;&#10;</xsl:text>
        </xsl:if>
      </xsl:if>

      <!-- check the out element -->
      <xsl:if test="$task/xslt/out"> <!-- out elements are optional -->
        <xsl:if test="$task/xslt/out/@var = ’true’">
          <xsl:if
test="not(contains($defined,concat(’&quot;’,$task/xslt/out,’&quot;’)))">
            <xsl:text>Error: Variable "</xsl:text>
            <xsl:value-of select="$task/xslt/out"/>
            <xsl:text>" referenced inside xslt/out task statement in task
"</xsl:text>
            <xsl:value-of select="$task/@taskid"/>
            <xsl:text>" has not been defined.&#10;&#10;</xsl:text>
          </xsl:if>
        </xsl:if>
      </xsl:if>
    </xsl:if>
    <!-- check var usage in wrapper element, if there is one -->
    <xsl:if test="$task/wrapper">
      <!-- check the wrapper input -->
      <xsl:if
test="not(contains($defined,concat('&quot;',$task/wrapper/wrapperInput,'&quot;'
)))">
        <xsl:text>Error: Variable "</xsl:text>
        <xsl:value-of select="$task/wrapper/wrapperInput"/>
        <xsl:text>" referenced inside wrapper/wrapperInput task statement in
task "</xsl:text>
        <xsl:value-of select="$task/@taskid"/>
        <xsl:text>" has not been defined.&#10;&#10;</xsl:text>
      </xsl:if>

      <!-- check the wrapper output -->
      <xsl:if
test="not(contains($defined,concat('&quot;',$task/wrapper/wrapperOutput,'&quot;
')))">
        <xsl:text>Error: Variable "</xsl:text>
        <xsl:value-of select="$task/wrapper/wrapperOutput"/>
        <xsl:text>" referenced inside wrapper/wrapperOutput task statement in
task "</xsl:text>
        <xsl:value-of select="$task/@taskid"/>
        <xsl:text>" has not been defined.&#10;&#10;</xsl:text>
      </xsl:if>
    </xsl:if>

    <!-- check var usage in outputVar element, if there is one -->
    <!-- *********** The inclusion of this task statement ************ -->
    <!-- *********** has not yet been confirmed           ************ -->

    <!-- check var usage in output element, if there is one -->
    <xsl:if test="$task/output">
      <!-- for each infoOut see if the var has been defined -->
      <xsl:for-each select="$task/output/infoOut">
        <!-- if the contents of the info out element are not defined -->
        <xsl:if
test="not(contains($defined,concat('&quot;',./infoName,'&quot;')))">
          <xsl:text>Error: Variable "</xsl:text>
          <xsl:value-of select="./infoName"/>
          <xsl:text>" referenced by output task statement in task "</xsl:text>
          <xsl:value-of select="$task/@taskid"/>
          <xsl:text>" has not been defined within the scope of that task.
&#10;&#10;</xsl:text>
        </xsl:if>
      </xsl:for-each>
    </xsl:if>
  </xsl:for-each>
</xsl:template>

<xsl:template name="checkWrappers">

  <!-- get a list of all of the defined wrappers -->
  <xsl:variable name="defined">
    <xsl:for-each select="$config//R[illegible]sourceName">
      <xsl:text>"</xsl:text><xsl:value-of select="./"/><xsl:text>"</xsl:text>
    </xsl:for-each>
  </xsl:variable>

  <xsl:for-each select="//wrapper/wrapperName">
    <xsl:if test="not(contains($defined,concat('&quot;',./,'&quot;')))">
```

```
      <xsl:text>Error: Wrapper "</xsl:text>
      <xsl:value-of select="./"/>
      <xsl:text>" referenced inside task "</xsl:text>
      <xsl:value-of select="../../@taskid"/>
      <xsl:text>" of task model "</xsl:text>
      <xsl:value-of select="$root/taskModel/@name"/>
      <xsl:text>" has not been defined inside the main configuration
file.&#10;&#10;</xsl:text>
    </xsl:if>
  </xsl:for-each>

</xsl:template>

<xsl:template name="checkAgents">

  <!-- get a list of all of the defined agent ids -->
  <xsl:variable name="defined">
    <xsl:for-each select="$config//agentId">
      <xsl:text>"</xsl:text><xsl:value-of select="./"/><xsl:text>"</xsl:text>
    </xsl:for-each>
  </xsl:variable>

  <xsl:for-each select="//agent">
    <xsl:if test="not(contains($defined,concat('&quot;',./,'&quot;')))">
      <xsl:text>Error: Agent "</xsl:text>
      <xsl:value-of select="./"/>
      <xsl:text>" referenced inside task "</xsl:text>
      <xsl:value-of select="../@taskid"/>
      <xsl:text>" of task model "</xsl:text>
      <xsl:value-of select="$root/taskModel/@name"/>
      <xsl:text>" has not been defined inside the main configuration
file.&#10;&#10;</xsl:text>
    </xsl:if>
  </xsl:for-each>


</xsl:template>



<!-- Helper functions -->

<!-- this template counts the number of a certain substring in the input
     string -->
<xsl:template name="count-substrings">
  <xsl:param name="text"/>
  <xsl:param name="substring"/>
  <xsl:param name="count" select="0"/>

  <!-- if no more of the substring quit recursion -->
  <xsl:if test="not(contains($text,$substring))">
    <xsl:value-of select="$count"/>
  </xsl:if>

  <!-- if more of the substring then continue and increment counter -->
  <xsl:if test="contains($text,$substring)">
    <xsl:call-template name="count-substrings">
      <xsl:with-param name="text" select="substring-after($text,$substring)"/>
      <xsl:with-param name="substring" select="$substring"/>
      <xsl:with-param name="count" select="$count + 1"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>

<!-- This template splits an external id along the | symbols -->
<xsl:template name="split-externalId">
  <xsl:param name="text"/>

  <xsl:if test="not(contains($text,'|'))">
    <part><xsl:value-of select="$text"/></part>
  </xsl:if>

  <xsl:if test="contains($text,'|')">
    <part><xsl:value-of select="substring-before($text,'|')"/></part>
    <xsl:call-template name="split-externalId">
      <xsl:with-param name="text" select="substring-after($text,'|')"/>
    </xsl:call-template>
  </xsl:if>

</xsl:template>

<!-- This template answers true if the supplied task id has been defined
     in the task model whose root has been passed in -->
<xsl:template name="isTaskIdDefined">
  <xsl:param name="id"/>
  <xsl:param name="rootOfTm"/>

  <xsl:variable name="all">
    <xsl:for-each select="$rootOfTm//@taskid">
      <xsl:text>"</xsl:text><xsl:value-of select="./"/><xsl:text>"</xsl:text>
    </xsl:for-each>
  </xsl:variable>

  <xsl:if test="contains($all,concat('&quot;',$id,'&quot;'))">
    <xsl:text>1</xsl:text> <!-- non empty string signifys true -->
  </xsl:if>
  <xsl:if test="not(contains($all,concat('&quot;',$id,'&quot;')))">
    <xsl:text>0</xsl:text>
  </xsl:if>

</xsl:template>


</xsl:stylesheet>
```